Enhancing Software Development Efficiency: An Automated Problem Frames to UML Transformation Method *

Hongbin Xiao
Guangxi Normal University
Bowen Zheng
China University of Petroleum
Simin Yang
Guangxi Normal University
Zhi Li
zhili@gxnu.edu.cn
Guangxi Normal University
Yilong Yang
Beihang University

Research Article

Keywords: Problem Frames, UML, Model-driven Engineering, Requirements Engineering

Posted Date: August 22nd, 2025

DOI: https://doi.org/10.21203/rs.3.rs-6977421/v1

Additional Declarations: No competing interests reported.

Enhancing Software Development Efficiency: An Automated Problem Frames to UML Transformation Method*

Hongbin Xiao [1], Bowen Zheng[3], Simin Yang[1], Zhi Li[1*], Yilong Yang[2*]

[1*]Key Lab of Education Blockchain and Intelligent Technology, Ministry of Education, Guangxi Normal University, Guilin, 541004,China.
[2]School of Software, Beihang University, Beijing, 100000,China.
[3]Qingdao Institute of Software, China University of Petroleum, Qingdao,266000, China.

*Corresponding author(s). E-mail(s): zhili@gxnu.edu.cn; yilongyang@buaa.edu.cn;
Contributing authors: hongbinxiao@stu.gxnu.edu.cn ;

Abstract

Modern software engineering faces a critical challenge in bridging the gap between requirements analysis and system design phases. Traditional methods lack systematic mapping mechanisms, leading to the disconnect from requirements design and inefficient transformation processes. This research addresses this challenge by establishing an automated mapping mechanism between problem frames and a unified modeling language. We propose the problem frames to a unified modeling language method, which combines Jackson's Problem Frames concept with model-driven engineering principles. The approach consists of three key components: an extended Problem Frames meta-model with enhanced semantic elements and transformation support structures, a graphical modeling platform developed using Eclipse Modeling Framework and Sirius following Meta-Object Facility three-layer architecture, and 24 transformation rules implemented using Atlas Transformation Language technology.Experimental validation on five representative open-source systems demonstrates significant improvements in software development efficiency. The method achieves a transformation accuracy of 93. 59%, representing an improvement 130% over the original approaches.

[0*]The PF2UML tool presented in this work has been accepted for demonstration at FSE'25 Tool Demo Track.

Practical implementation shows an average time reduction of 78.64% in end-to-end modeling processes, with a theoretical return on investment of 170. 8% and a practical average return on investment of 236.6%. The platform automatically generates four types of UML diagrams: conceptual class diagrams, use case diagrams, service diagrams, and system sequence diagrams. The PF2UML method successfully establishes seamless integration from requirements modeling to system design, enhancing traceability while reducing manual expertise dependence and significantly improving consistency for complex interdisciplinary system development.

Keywords: Problem Frames, UML, Model-driven Engineering, Requirements Engineering

1 Introduction

The software development lifecycle is the core framework of modern software engineering, encompassing the entire process from requirements analysis to system design, implementation, and maintenance[1]. In this process, the requirements phase and the system design phase serve as critical links that ensure system success, and the quality of their connection directly impacts the ultimate success of software products[2, 3]. With increasing system complexity and cross-disciplinary collaboration needs, the gap between traditional requirements analysis and system design methods has become increasingly prominent[4, 5].

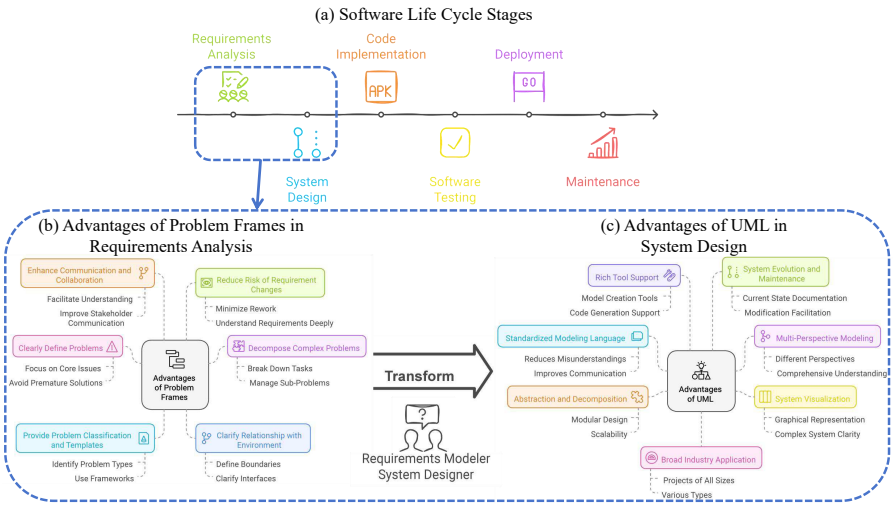


Fig. 1 Software lifecycle stages and the respective advantages of Problem Frames in requirements analysis and UML in system design phases.

Problem Frames (PF), as a structured requirement modeling method, provides a clear framework for complex requirements through explicit domain boundaries and phenomenon linkages[6]. Meanwhile, Unified Modeling Language (UML), as an industry standard for the system design phase, is widely applied to describe system functionality, structure, and behavior[7]. Fig. 1 illustrates the stages of the software life cycle and presents the respective advantages of Problem Frames and UML in the requirements analysis and system design phases. However, there is a lack of systematic mapping mechanisms between these two methods, leading to issues such as disconnection of the requirements design, inefficient transformation, and insufficient consistency traceability[8–10].

Existing research focuses mainly on single aspects, such as the modeling of requirements or design optimization, neglecting the transformation rules and the support of the tool between them[11]. This makes it difficult to fully utilize high-quality models from the requirements phase in the design phase. This research gap not only reduces software development efficiency but also increases the risk of requirements deviation, affecting the quality of the final product.

This research proposes the Problem Frames to Unified Modeling Language (PF2UML) method, which aims to establish an automated mapping mechanism between Problem Frames and UML. The method first builds an extended Problem Frames meta-model and a problem-oriented requirements modeling platform based on Jackson's Problem Frames concept combined with model-driven engineering principles. The extended meta-model introduces semantic enhancement and mapping support elements, including foundational elements (inherited from original PF), enhanced core elements with semantic extensions, and newly introduced support structures for transformation. Then, by analyzing the semantic relationships between the Problem Frames meta-model and the UML meta-model, it designs comprehensive model transformation algorithms using ATL (Atlas Transformation Language) to achieve seamless integration from requirements modeling to system design.

The main contributions of this paper include three aspects:

1. **Extended Problem Frames Meta-model with Semantic Mapping** We propose an extended Problem Frames meta-model that enhances Jackson's framework with semantic annotations for UML transformation. The extension includes foundational elements, enhanced core elements with transformation attributes , and new support structures. We design 24 transformation rules enabling automated mapping from Problem Frames to UML model.
2. **Model-Driven Requirements Modeling Platform** We develop a graphical modeling platform using Eclipse EMF and Sirius, following MOF three-layer architecture. The platform provides a complete toolchain from meta-model construction to visual modeling, featuring drag-and-drop operations and structured requirements modeling for complex interdisciplinary systems.
3. **PF2UML Automated Transformation with Validation** We present the PF2UML transformation method using ATL technology. Experimental validation shows a theoretical ROI of 170. 8%, a practical ROI of 236. 6%, a transformation accuracy of 93. 59% (130% improvement) and a time reduction of 78. 64% in

modeling processes, demonstrating significant efficiency gains in automated model transformation.

The remainder of this paper is organized as follows. Section 2 delineates the core concepts utilized herein. Section 3 delves into the work related to our study. In Section 4, we describe the research related to PF2UML in detail. Section 5 presents the practicality of model transformation through a case study encompassing six open-source systems as a proof of concept. Section 6 evaluates the proposed methodology based on the findings of the case study. In conclusion, Section 7 outlines the conclusions drawn and proposes directions for future research.

2 Foundational Theories and Technologies

This section introduces the foundational theories and technologies that our research work relies upon, establishing the groundwork for in-depth discussions in subsequent sections. First, we will explore Problem-Oriented Requirements Modeling as a structured requirements analysis method, examining its core concepts and application value; second, we will introduce the Unified Modeling Language and its key role and primary graphical representations in system design; finally, we will elaborate on the basic principles of Model-Driven Engineering (MDE) and its applications in software development. Through a systematic review of these three domains, we will establish a theoretical bridge from requirements analysis to system design, providing technical support to address the transformation challenges between them.

2.1 Problem Frames Preliminary Knowledge

Problem Frames is a systematic software requirements analysis and system design methodology proposed by Michael Jackson[6]. This approach employs graphical modeling techniques to clearly describe the complex relationships between software systems and their operating environments. By systematically decomposing complex software problems into different domains and precisely defining the interactions between these domains, Problem Frames helps developers gain deep insights into the essential characteristics of problems and system boundaries, thus establishing a solid foundation for the subsequent design and implementation of the system[12]. Problem Frames utilizes a standardized graphical notation system to represent different components of the system, primarily including the following core elements:

- **Machine Domain** : Represents the software system to be developed, serving as the core implementation component of the problem solution.
- **Problem Domains** : Based on their nature and behavioral characteristics, they are further subdivided into three subcategories:
  - **Causal Domains**: Entities capable of executing concrete actions and exerting causal influences.
  - **Biddable Domains** : Domains involving human user participation, whose behavior exhibits unpredictability.

– **Lexical Domains**: Domains specialized in data storage, information representation, and symbolic operations.

- **Requirement Domain**: Defines the collection of functional and non-functional requirements that the system must satisfy.
- **Interface** : Represent shared phenomena existing between different domains, defining direct interaction interfaces and communication methods between domains.
- **Requirement Reference**: Represent general requirement reference relationships, illustrating how specific requirements involve and affect related problem domains.
- **Constraining Requirement Reference**: Represent requirement references with constraining nature, explicitly specifying mandatory constraints and restrictions that requirements impose on system behavior.

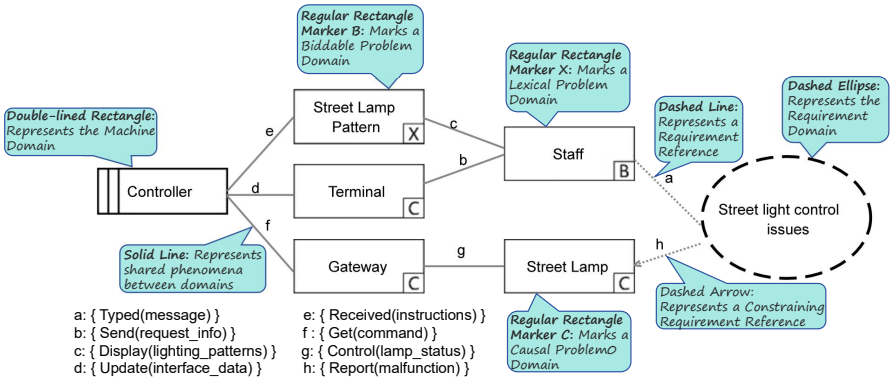


Fig. 2 Problem Frames Modeling Diagram for Street Lighting Control System

The street lighting control system problem frames depicted in Fig. 2. It comprises six principal domains: The Controller functions as the machine domain, constituting the core component of the software system under development; Staff represents a biddable domain, characterized by human operators exhibiting inherently unpredictable behavioral patterns; Terminal, Gateway, and Street Lamp constitute causal domains, responsible for user interface management, communication mediation, and illumination control, respectively; Street Lamp Pattern serves as a lexical domain, managing the data storage and representational aspects of lighting configurations. The requirement domain "Street light control issues" is interconnected with relevant domains via dashed connectors, demonstrating the cross-domain characteristics of system requirements, wherein the dashed arrow directed toward Street Lamp signifies mandatory behavioral constraints imposed upon the lighting infrastructure. The system architecture defines inter-domain interactions through eight distinct shared phenomena: staff personnel input control directives via the terminal interface (a), the terminal transmits request information to the controller (b,d,f), the controller dispatches control commands to street lamps through the gateway infrastructure (e,g), street lamps

execute control operations and provide status feedback (h), while the system concurrently displays lighting pattern information (c), thereby establishing a comprehensive monitoring and control feedback loop.

This problem frames analysis methodology provides systematic clarification of system boundaries and component responsibilities, establishes robust requirements traceability mechanisms, provides structured architectural guidance for system design, and functions as an effective stakeholder communication framework to facilitate shared understanding of system complexities and interdependencies.

2.2 Unified Modeling Language

Unified Modeling Language (UML) [1] is a standardized modeling language used for the analysis, design, and documentation of software systems. Provides a set of graphical notations and rules for describing and denoting the structure, behavior, and interactions of software systems, which aid developers in the analysis, design, and documentation during the software development process. The output model in this paper is a lightweight formal requirements model [13–15], which includes the following components.

- **Conceptual Class Diagrams**: Class diagrams represent the types, attributes, and relationships of objects within a system. They depict classes, interfaces, associations, and inheritance within the system, serving to describe its structure and data requirements.
- **Use Case Diagrams**: These diagrams articulate the functional requirements of a system, illustrating interactions between the system and external actors. They demonstrate the relationships between actors and use cases, aiding in the identification of system functionalities and user requirements.
- **Service Diagrams**: A Service Diagram is a graphical representation used to describe services and their interactions within a Service-Oriented Architecture (SOA). It is a structured diagram that visualizes and analyzes the services within a system and the relationships and communications between them.
- **System Sequence Diagrams**: System sequence diagrams detail the order of interactions and message passing between objects in a system. They show the flow of messages and the process of interaction between objects, describing the behavior requirements and temporal relationships of the system.

2.3 Model-Driven Engineering

Model-Driven Engineering (MDE) emphasizes the strategic use of formalized model representations to systematically navigate software and system development, where models ranging from abstract conceptual frameworks to formal specifications precisely delineate the structure, behavior, and interactions of the system [16, 17]. Unlike conventional code-centric methodologies, MDE provides an intuitive, visually oriented paradigm that operates through hierarchical abstraction layers, utilizing metameta models such as the Meta-Object Facility (MOF) and Ecore within the Eclipse Modeling

[1] UML https://www.omg.org/spec/UML/2.5.1/PDF

Framework (EMF) to enable the creation of models and the specification of metamodels [18]. At its core, model transformation, a fundamental MDE technique, enables developers to navigate between different abstraction levels and perspectives through vertical transformations (design requirements, design-to-implementation, platform independent to platform-specific models) and horizontal transformations between viewpoints (structural, behavioral, temporal perspectives) [19, 20]. This integrated approach addresses the limitation that single models often do not fully capture all aspects of the system and levels of complexity [16], facilitating development processes by maintaining systematic traceability of conceptual models of high level through concrete code implementations via systematic model transformations, thereby allowing comprehensive representation of the system while managing complexity through appropriate abstraction mechanisms [21, 22].

- EMF is an open source framework provided by Eclipse for building, managing, and manipulating modeled data [23]. It stands out as a robust and widely adopted framework designed to facilitate the creation, management and manipulation of modeled data [24]. EMF equips developers with an extensive toolkit and library, fostering model-driven software development and encouraging the generation and application of domain models. Its application spans a multitude of disciplines, such as software engineering, system modeling, data management, and simulation, showcasing its versatility and utility [25].
- Sirius [2] is another open-source project in Eclipse that focuses on the rapid development of bespoke graphical editors. It provides a straightforward and effective approach to define and craft domain-specific visualization tools, enhancing visual editing and interaction capabilities with domain models [26]. Sirius enables developers to effortlessly construct custom graphical editors based on the unique requirements of specific domains [27]. Suitable for diverse sectors such as software engineering, systems engineering, and business process modeling, Sirius provides a versatile and scalable platform. This platform empowers developers to visually create and manage domain models, significantly improving development productivity and the expressive power of model visualizations.
- ATL (Atlas Transformation Language) technology[28], is utilized to define and execute transformation rules between models. Developed on the principles and practices of MDE, ATL offers a declarative approach to describe transformation rules. It uses a syntax similar to rules and pattern matching, enabling developers to define mappings and transformation operations between source and target models [29].

3 Related Work

This section reviews related research from two core dimensions: problem-oriented requirements modeling and requirements transformation. As shown in Fig. 3, related research mainly covers two major aspects: modeling tool development (including non-model-driven and model-driven approaches) and requirements transformation techniques (including transformation to UML and non-UML models).

[2]Sirius https://eclipse.org/sirius/

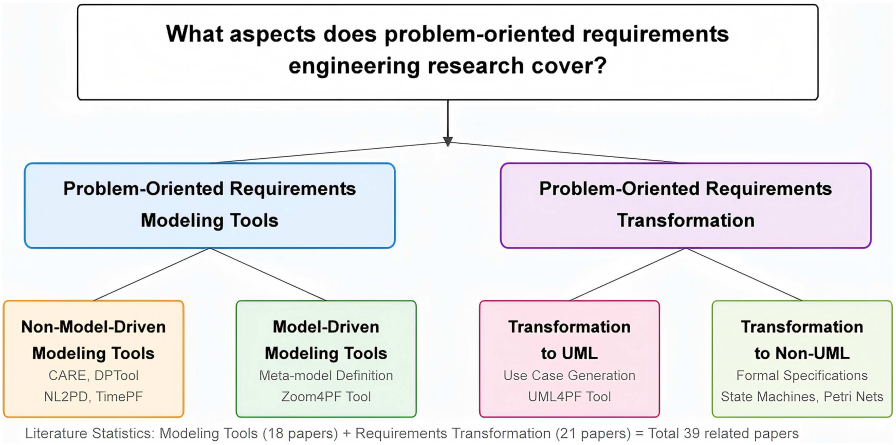


Fig. 3 Classification Framework of Problem-Oriented Requirements Engineering Research

3.1 Problem-Oriented Requirements Modeling

Problem-oriented requirements modeling tools aim to support requirements engineering activities through structured problem analysis approaches. Based on different modeling methodologies, these tools can be categorized into two types: non-model-driven and model-driven approaches.

Non-model-driven problem-oriented requirements modeling primarily focuses on practicality and usability through direct graphical interfaces and automation. Basic tools like CARE and DPTool provide intuitive problem diagram creation and simplified workflows[30, 31]. Advanced tools include NL2PD for automatic natural language to problem diagram generation, and Zheng et al.'s multimedia approach for richer problem expression[32, 33]. Specialized tools have been developed for specific scenarios: TimePF[34] for temporal requirements, PF4MD[35] for microservice decomposition, Trace4PF[36] for traceability management, and RE4CPS[37] for cyber-physical systems. These tools collectively improve the practical application of problem frames methods in software development[30–37].

Model-driven problem-oriented requirements modeling establishes formal meta-models to support the modeling and analysis of systematic problem frames. The models include Hatebur et al.'s UML-based syntax and semantics definition[38], Colombo et al.'s extended meta-model for complex problem decomposition[39–41], and Lavazza and Coen-Porisini's modular modeling approach[42, 43].Domain-specific extensions include Yang's[44] meta-model for metaverse systems with VR/AR support, and Han et al.'s [45]integration of goal-oriented requirements with problem frames for adaptive cyber-physical systems. Implementations of tools such as Zoom4PF[46], OpenArgue[47], and PF-HCPS[48] demonstrate practical applications of meta-model-driven approaches, providing theoretical foundations and tool support for formal problem frames applications.

Non-model-driven approaches[38–48] offer ease of use and low learning barriers through intuitive interfaces but lack formal foundations for model consistency and verification. Model-driven approaches provide formal meta-model-based validation and

automation, but have high learning costs and complexity. Both approaches share a critical limitation[30–48]: the lack of smooth transition mechanisms from requirements analysis to system design, since problem frames models cannot be directly converted to UML design models, creating a gap that affects development efficiency.

Therefore, this research will extend the problem frame meta-model to establish mapping rules and automated transformation mechanisms from problem frames to UML models, achieving seamless integration between requirements engineering and system design to improve overall software development efficiency and quality.

3.2 Problem-Oriented Requirements Transformation

Problem-oriented requirements transformation research focuses on converting problem-oriented requirements models into design models or implementation code, serving as a bridge between requirements engineering and software design. Methods are categorized into UML and non-UML transformation approaches.

UML transformation method includes Choppy and research has established various connections between problem frames and UML modeling. Key contributions include Reggio's [49] systematic mapping rules for transforming problem frame elements into UML diagrams, and Lavazza et al.'s [50, 51] integrated approaches combining problem frames analysis with UML design. Tool support has been developed through UML4PF [52], which provides automated transformation with graphical interfaces and consistency check. Domain-specific applications include security requirements modeling [53] and smart grid trust relationship modeling using specialized UML profiles.

Non-UML transformation methods encompass formal specifications, state machines, Petri nets, and generation of test code. Key approaches include Rapanotti et al.'s [54] problem reduction for specification derivation, Li et al.'s [55]complete transformation frameworks, and B-method mapping constructs. Other methods cover state machine generation through phenomena analysis, Petri net transformation for controller design[56], SysML conversion for complex systems, and Gao's automated test code generation[57, 58]. Specialized tools include DRAP-PF[59] for reusable analysis patterns and Li Zhi's [60] requirement-to-specification transformation. UML methods offer standardization for object-oriented development, while non-UML approaches provide flexibility for formal verification, concurrent modeling, and domain-specific applications.

Existing requirement transformation methods are categorized into UML and non-UML approaches. UML transformation [49–53] generates use case diagrams, class diagrams, and sequence diagrams with high standardization and tool support for object-oriented development. Non-UML transformation [54–60] targets formal specifications, state machines, Petri nets, and test code, offering diverse options for formal verification, concurrent modeling, and domain-specific applications. However, existing methods suffer from coarse transformation granularity and rigid transformation rules, hindering fine-grained semantic preservation and flexible automation.

To address these limitations, this paper proposes a Problem Frames to Unified Modeling Language transformation method. The approach analyzes semantic

relationships between Problem Frames and UML meta-models to design transformation algorithms, achieving more precise, flexible, and automated requirement transformation.

4 PF2UML

To facilitate effective collaboration between domain experts and software engineers in complex system development, we propose PF2UML[3], a model-driven transformation method that converts Problem Frames models into comprehensive UML models. Our approach extends the Problem Frames meta-model using Model-Driven Engineering principles and employs Sirius for graphical representation of model elements. As illustrated in Fig. 4, the PF2UML transformation process takes a Problem Frames model as input, collaboratively created by domain stakeholders and software engineers, and applies ATL-based transformation algorithms with logical controls to process key elements, including problem domains, shared phenomena, requirement domains and attribute domains. The algorithm automatically generates four types of UML diagram as output, providing a seamless bridge from requirements modeling to system design. The following sections detail our meta-model extensions, graphical notation system, and transformation algorithms to provide a comprehensive understanding of the PF2UML methodology.

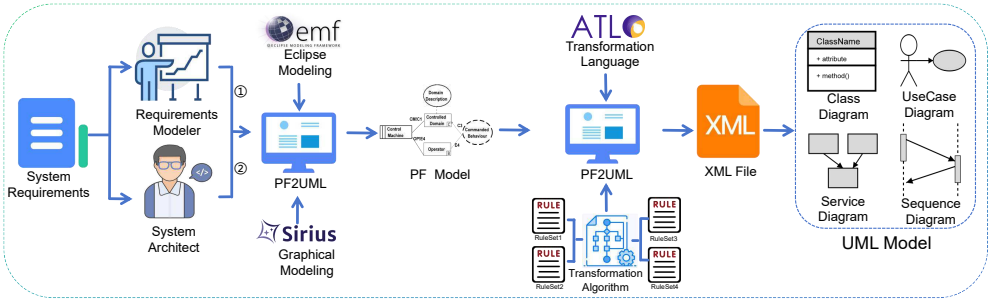


Fig. 4 PF2UML Overview.

4.1 Extension of Problem Frames meta-model for UML Transformation

In order to realize the automated conversion mechanism from PF to UML, the isomorphism between the two types of models in terms of structural representation and semantic dimensions needs to be solved first. Based on Jackson's original PF approach, this study attempts to redesign and extend its meta-model by combining semantic mapping relationships and structural alignment requirements, and initially constructs a structural framework suitable for model-driven engineering approaches as a feasible path for exploring the conversion of PF models to UML models. Fig. 5 illustrates the

[3]https://github.com/Hongbin-Xiao/PF2UML

structure of the extended meta-model PF constructed in this study. Compared with the traditional PF model, the extended model introduces several semantic enhancement and mapping support elements for the interface with the UML semantic structure on the basis of maintaining the original modeling paradigm. The general extension structure can be divided into the following three functional areas.

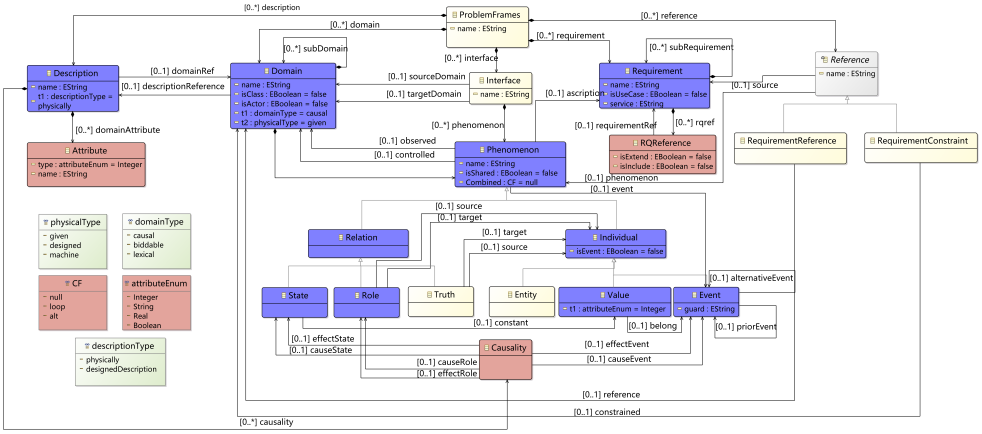


Fig. 5 The Extended Meta-model of Problem Frames.

Foundational meta-model Elements (Yellow): The yellow elements in the extended meta-model represent the foundational components inherited from the original Problem Frames approach. These include classes such as problem frames, interface, reference, requirement reference, requirement constraint, truth, and entity. Together, they define the static structure and semantic backbone of the model. Problem Frames serves as the container for organizing requirements, domains, and interfaces. Reference acts as an abstract superclass, enabling a consistent structural basis for modeling requirement relations and constraints. These elements preserve the essential modeling paradigm of PF and provide the structural basis for traceability and transformation toward UML models.

Enhanced Core Elements with Semantic Extensions (Blue): The blue elements represent the original PF model classes that have been extended with additional attributes to support semantic annotation and transformation readiness. This includes domain, requirement, phenomenon, relationship, state, role, value, event, individual, and description. For example, Domain now includes properties such as isClass and isActor, while Requirement supports isUseCase and service to aid in use case generation. The phenomenon is augmented with shared flags and control attributes such as Combined, and Event allows guarded transitions via guard. These extensions enable the model to capture not only structural properties, but also behavioral semantics needed for mapping to UML class, use case, and sequence diagrams.

Newly Introduced Support Structures for Transformation (Red): The red elements are newly introduced in this study to enhance the model's ability to support the transformation of PF2UML. These include structural causality, which models

causal dependencies between events, states, and roles, and RQReference. The attribute enables detailed attribute modeling for domains, facilitating field definition in UML. Additionally, several types of enumeration are defined: CF (e.g. null, loop, alt) for control fragments, attributeEnum (e.g. inte.g.er, real, boolean) for data typing, among others. These additions serve to constrain, standardize, and enrich the semantics of the meta-model, laying the groundwork for automated, type-safe, and rule-based UML model generation.

Through the proposed extensions and enhancements, the Problem Frames meta-model aims to improve its semantic expressiveness and modeling precision, while maintaining the core structure of the original framework. These modifications are designed to better support the representation of multi-domain requirements in complex systems and to provide a conceptual basis for establishing a semantic link between requirements modeling and system design. In the context of the transformation of the PF2UML model, the extended structure introduces more detailed attribute specifications and logical constraints, which are intended to enable a structured and traceable representation of requirements, thus facilitating more consistent and potentially more effective model translation.

Fig.6 shows the Problem Frames example model of the CoCoME system constructed based on this extended meta-model in this study. The model demonstrates the comprehensive structural representation capabilities of the extended meta-model, including various element types such as Problem Domains, Requirement Domains, Shared Phenomena, Interface Instances, and their associated attributes and relationships. However, the current modeling approach still relies on the implementation of the underlying modeling language, which is complicated in operation and weak in visibility, and is not conducive to the understanding and participation of nontechnical users in the modeling process. Therefore, the next section describes the graphical modeling platform constructed in this study, which implements visual modeling operations based on this PF extended meta-model to enhance the user experience, modeling efficiency, and the practicality of subsequent model transformation.

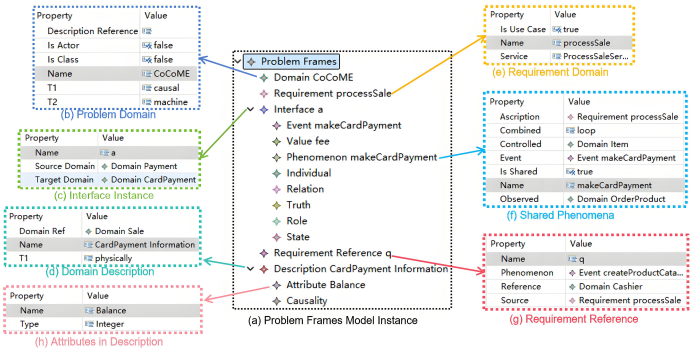


Fig. 6 Extended Problem Frames meta-model Instance Example

4.2 Problem-oriented modeling platform for model transformation

In order to realize the automatic conversion from PF models to UML models, it is crucial to build a modeling system with a standardized structure, consistent semantics, and Model-Driven Engineering specifications. In this study, under the guidance of MDE methodology, we adopt the MOF three-layer modeling architecture to design and implement the whole process of modeling, graphical visualization, and model transformation of PF models [18].

As shown in Fig.7, the system follows the typical MOF three-layer structure: M1 layer is a concrete model instance, such as a .pf file; M2 layer is the meta-model that defines the language, including the PF meta-model and the UML meta-model; M3 layer is the meta-modeling language MOF itself. By establishing semantic mapping rules between the M2 layers and with the help of ATL transformation engine, structural mapping and semantic preservation from PF instance models to UML models can be realized.

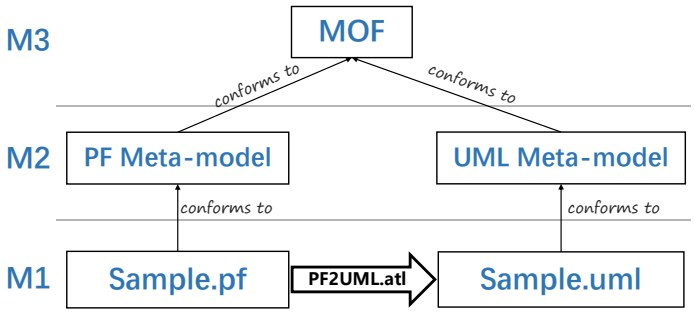


Fig. 7 Model Transformation Architecture from PF Meta-model to UML Meta-model Using MOF.

In order to ensure the structural legitimacy and semantic consistency of the M1 layer PF model, it is necessary to construct a modeling platform that supports PF meta-models as a graphical modeling front-end and also as a semantic input source for the PF2UML model transformation process. In this study, a graphical PF modeling platform based on Eclipse Modeling Framework (EMF) with Sirius plug-in is constructed.

Fig. 8 shows the implementation process of the platform. First, the core concepts in Problem Frames theory (e.g., Machine, LexicalDomain, Requirement, etc.) are constructed at the semantic layer, and the construction of.ecore meta-models and.xmi model instances is completed in EMF. Subsequently, the graphical representation model (.odesign) is defined through Sirius and the visual modeling file (.aird) is generated, which enables the graphical construction, interactive editing and property configuration of the PF model.

In order to ensure that the modeling results of the platform are aligned with the structure of the PF meta-model, and at the same time have good UML mapping capability, the platform introduces a fine-grained node and relationship expression

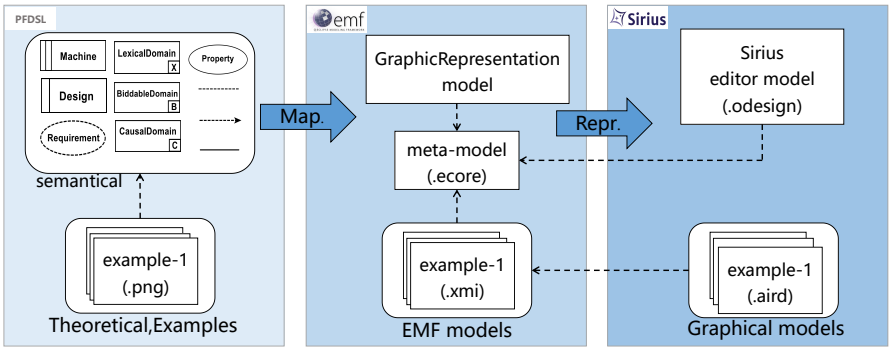


Fig. 8 Workflow for Problem Frames Meta-model Implementation and Visualization Using EMF and Sirius.

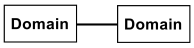
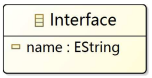
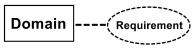
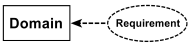
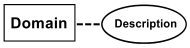
mechanism at the graphical mapping level. Table 1 shows the graphical representation, meta-model definition, and semantic description of the elements 'nodes' in the platform, including key components such as Machine, Requirement, Causal Domain, Lexical Domain, etc., which are bound to the corresponding semantics in the meta-model PF and labeled with the type field (e.g. physicalType). fields (e.g. physicalType, domainType, etc.) for subsequent transformation calls.

Table 1 Description of nodes containing multimedia elements

Graphical	Meta-Model Element	Descriptions
Machine	Domain - name : EString - t2 : physicalType = machine	The machine is where the software to be run.
Designed Domain	Domain - name : EString - t2 : physicalType = designed	Lexical domains provide physical space where the data is stored, thus with some causality in their storage behaviors.
Causal Domain C	Domain - name : EString - t2 : physicalType = designed	Causal domains contain predictability in their behaviors.
Biddable Domain B	Domain - name : EString - t1 : domainType = biddable	biddable domains (usually human beings) do not contain predictability in their behaviors.
Lexical Domain X	Domain - name : EString - t1 : domainType = lexical	A lexical domain is a physical representation of data.
Requirement	Requirement - name : EString - isUseCase : EBoolean = false - service : EString	They describe the behavioral references or constraints, prescribed by the problem owners.
Description	Description - name : EString - t1 : descriptionType = physically	They describe internal or external causal relationships or other characteristics of the domain.

In terms of relationship construction, the modeling of "edges" in the platform also strictly corresponds to the PF semantic structure. Table 2 shows the graphical symbols, connection objects, and semantic functions of common relationships in the platform, including Interface, RequirementReference, RequirementConstraint, and DescriptionReference. These graphical relationships will be directly assigned to the associative relationships, extension mechanisms, and behavioral dependency paths of UML in subsequent model transformations, constituting a one-to-one correspondence between the modeling structure and UML semantics.

Table 2 Inter-Node Relationship Notations and Descriptions

Graphical	Edge	Source	Target	Descriptions
		Domain	Domain	A solid line connecting two domains is an interface of shared phenomena.
		Domain	Requirement	A dashed line connecting a domain and a requirement is a requirement reference.
		Domain	Requirement	A dashed arrow connecting a domain and a requirement is a constraining requirement reference.
		Domain	Description	A dashed line connecting a domain and a description is a description reference.

The platform interface, shown in Fig. 9, provides a complete Problem Diagram drawing function, including four main functional areas: Menu Bar & Toolbar, Modeling Elements Panel, Modeling Canvas, and Multi-Panel Workspace. The Menu Bar & Toolbar offers comprehensive file operations, editing functions, view options, and modeling tools. The Modeling Elements Panel provides various Problem Frames modeling elements such as domain elements, requirement symbols, and interface connectors that users can access through drag-and-drop operations. The central Modeling Canvas serves as the primary visual modeling workspace where users can create and connect Problem Frames diagrams. The Multi-Panel Workspace integrates multiple auxiliary functions, including Properties Editor for attribute configuration, Model Explorer for navigation, Outline View for structural overview, and Problems Panel for model validation. For detailed usage examples and practical applications of these features, please refer to Section 5.2 Requirements Modeling.

The platform helps to ensure that the exported .xmi model conforms to the structural specifications of the PF meta-model and can be directly imported into the PF2UML.atl conversion engine. This modeling platform aims to provide an intuitive and structured environment for building problem frames models while facilitating seamless integration between graphical elements and the underlying meta-model. Beyond serving as a graphical modeling tool, the platform attempts to function as an effective intermediary layer that bridges requirements modeling and system design within a model-driven engineering context.

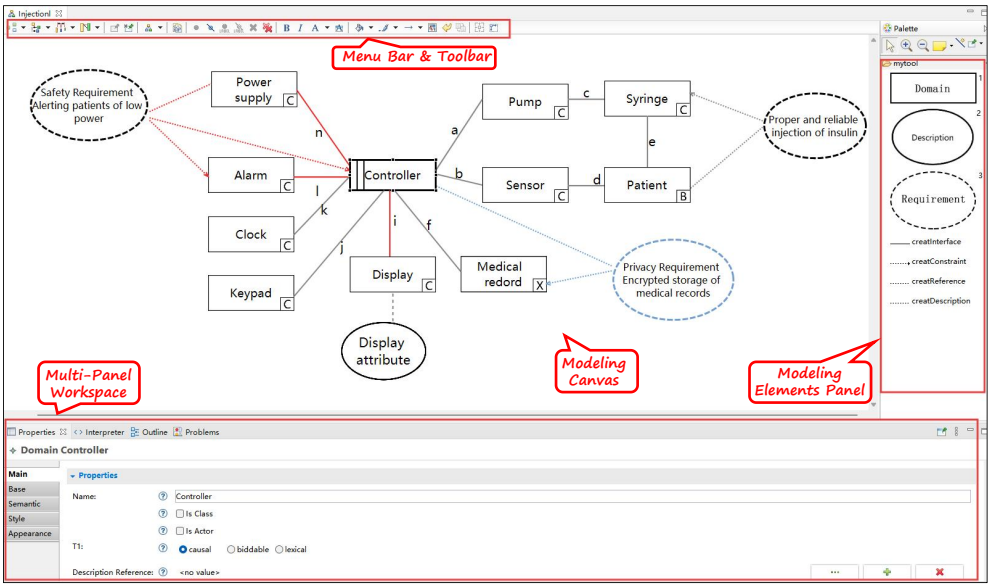


Fig. 9 Problem Frames Requirements Modelling Platform

4.3 Transforming Problem Models Into UML Models

In this section, we present a detailed methodology for transforming problem models into UML models. The process begins with an analysis of the UML metamodel and is divided into four focused modules: PF2ConceptualClassDiagram, PF2UseCaseDiagram, PF2ServiceDiagram, and PF2SystemSequenceDiagram (see Fig. 10). For each module, we define the corresponding input and output mappings[4]. Based on the foundational semantics of both Problem Frames and UML, we formulate transformation rules adapted to the characteristics of each diagram type. These rules are organized logically and rely on precise element selection to support semantic consistency. Together, they constitute a transformation algorithm designed to guide the conversion from problem models to UML representations. The transformation logic and rule definitions for each module are described in detail to enhance clarity and ensure technical soundness.

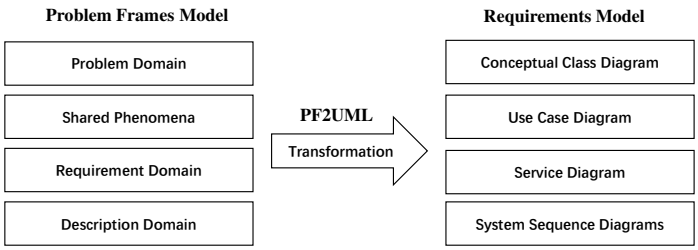


Fig. 10 Transformation Process from Problem Frames Model to UML Requirements Model

[4] https://htmlpreview.github.io/?https://github.com/Hongbin-Xiao/PF2UML/blob/main/PF-UMLMapping.html

4.3.1 Target UML Meta-Models Overview

To ensure a consistent and semantically accurate transformation from Problem Frames models, it is essential to first clarify the structure of the UML meta-models that serve as the transformation targets. In this subsection, we provide an integrated overview of the four UML meta-model segments involved in our transformation process: the Conceptual Class Diagram, the Use Case Diagram, the Service Diagram, and the System Sequence Diagram. Each of these diagrams plays a distinct role in capturing the structure, functionality, service behavior, and dynamic interactions of the system. Fig.11 illustrates the target meta-model: the UML meta-model.

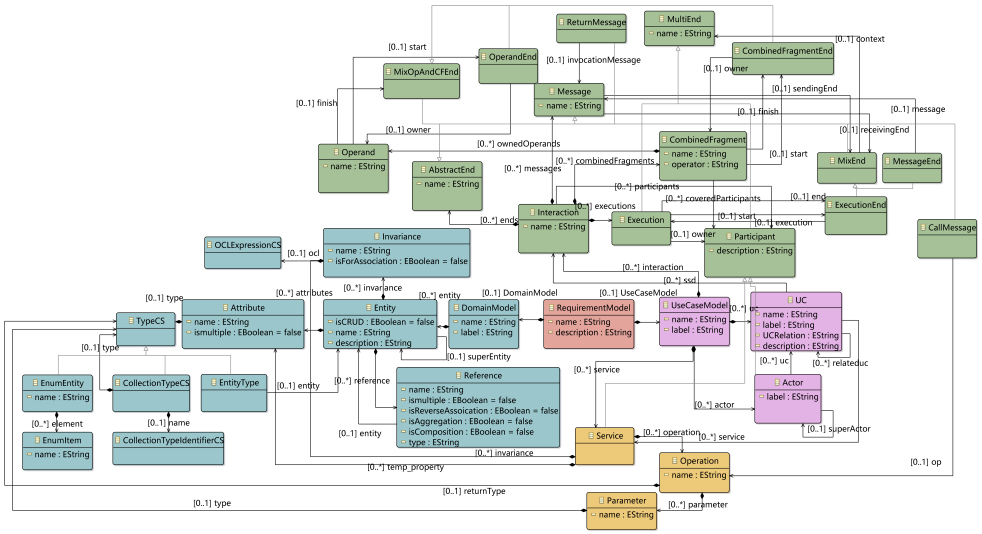


Fig. 11 The Meta-model of UML.

Conceptual Class Diagram Meta-model (Blue): This segment defines the comprehensive DomainModel, which encompasses multiple Entity instances that represent domain classes. Each entity incorporates attributes (Attribute) and supports object-oriented inheritance mechanisms via superEntity relationships. Inter-entity associations are systematically modeled through reference constructs, effectively capturing complex aggregation, composition, and multiplicity relationships. Attribute typing is rigorously defined using TypeCS, which includes CollectionTypeCS, EntityType, and EnumEntity for a comprehensive type specification.

Use Case Diagram Meta-model (Purple): The UseCaseModel is architecturally composed of Actor and UC (Use Case) instances that define the system functionality and user interactions. Actors may establish inheritance hierarchies through superActor relationships and are systematically linked to use cases via well-defined referential relationships. The use cases themselves can be interconnected using the UCRelation property, which elegantly captures behavioral dependencies including include and extend relationships.

Service Diagram Meta-model (Orange):A Service represents a cohesive functional unit that encapsulates Operation elements, each of which may contain Parameter and Attribute instances. All attributes and parameters are strongly typed through TypeCS, ensuring safety and consistency of type. This meta-model provides a structured foundation for formalizing system-level functionality in service-oriented architectures.

System Sequence Diagram Meta-model (Green): This meta-model is centered around the interaction element, which orchestrates and aggregates key behavioral components including Participants (encompassing both Actors and Services), Execution contexts, CallMessage and ReturnMessage exchanges, CombinedFragment constructs, and various abstract communication endpoints. It comprehensively captures the dynamic run-time behavior and intricate interaction patterns among distributed system components.

The UML meta-model serves as the foundation for our subsequent transformation rules, ensuring that structural, functional, and behavioral elements from the Problem Frames model are accurately mapped into standardized UML representation.

4.3.2 PF2ConceptualClassDiagram

In the Problem Frames model, a Domain represents an abstract conceptual unit, essentially modeling a physical or logical component of the system environment. This aligns closely with the entity in the UML class diagrams, which represents a class comprising attributes and relationships. Each domain may include descriptive properties, termed domainAttributes, which naturally correspond to UML Attribute elements. These capture intrinsic characteristics of the domain and are directly mapped to class-level properties in UML.Moreover, the Relation elements in the problem diagram, describing interactions or dependencies between domains, can be mapped to reference elements in UML. These include various forms of associations such as aggregation (isAggregation), composition (isComposition), and bidirectional relationships(isReverseAssociation). To more visually illustrate the conversion process of the Problem Frames model to the UML class diagram, Fig.12 shows the mapping path between key elements.

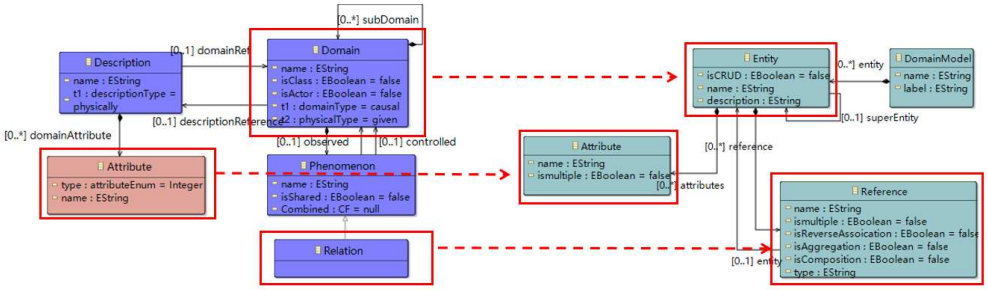


Fig. 12 Mapping Paths of Domain, Attribute, and Phenomenon from PF to UML Class Diagram

The rules for transforming Problem Frames models into the class diagram portion of UML models are as follows:

$$R_1 : \frac{\text{PF! PFModel(name, domain)}}{\text{RE! DomainModel(name, entity)}} \tag{1}$$

R_1 , defines how to transform from the top-level structure of the problem frames model to the UML domain model.

$$R_2 : \frac{\text{PF!Domain(name,pfattribute,PF!Relation)}}{\text{RE!Entity(name,reattributes,reference)}} \tag{2}$$

R_2 , is responsible for transforming domains within Problem Frames into UML entities, handling the attributes and relationships of the domains.

$$R_3 : \frac{\text{PF!Attribute(name,pfAttr.type.toString)}}{\text{RE!Attribute(name,type)}} \tag{3}$$

R_3 , transforms the attributes from Problem Frames into attributes of UML entities, including the nuances of the type transformation.

$$R_4 : \frac{\text{PF!Relation(name,true,RE!Entity)}}{\text{RE!Reference(name,isAggregation,entity)}} \tag{4}$$

R_4 , addresses the transformation of relationships within Problem Frames into UML reference relationships, which involves the identification of aggregation relationships.

4.3.3 PF2UseCaseDiagram

Before introducing the transformation rules for the PF2UseCaseDiagram, it is important to clarify the semantic correspondence between the elements in the Problem Diagram and those in the UML Use Case Diagram. In the Problem Frames model, a Domain typically represents an external entity or environment that interacts with system requirements. In the UML Use Case Diagram, an Actor plays a similar role: It models external users or systems that initiate or participate in use case execution. Likewise, a Requirement in the problem model describes a functional goal of the system, which aligns directly with a Use Case in UML, representing a specific service or functionality provided by the system to an actor. Reference relationships between domains and requirements form the basis for actor–use case associations, while connections between requirements can be mapped to UML that include and extend relationships. This semantic alignment provides a solid foundation for designing consistent and accurate transformation rules.To further illustrate the semantic mapping relationship between the Problem Frames model and the UML use case diagram, Fig. 13 visualizes how the Domain and Requirement elements are transformed into the actor and use case, and how their interactive associations are preserved in the target model.

The Problem Frames model to UML model transformation for the Use Case diagram part comprises three transformation rules:

$$R_5 : \frac{\text{PF!PFModel(name,domain,requirement)}}{\text{RE!UseCaseModel(name,actor,uc,service,interaction)}} \tag{5}$$

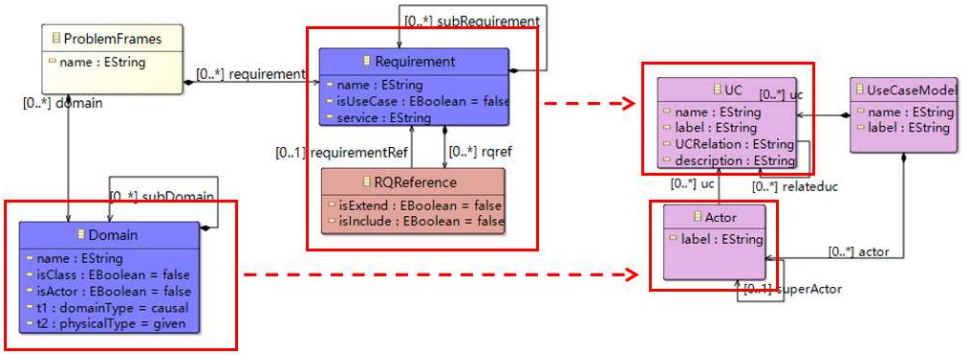


Fig. 13 Mapping Paths of Domain and Requirement from PF to UML Use Case Diagram

R_5 is responsible for transforming the Problem Frames model into the Use Case model. Maps the names, domains, and requirements of Problem Frames to the names, actors, use cases, services, and interactions in the Use Case model.

$$R_6 : \frac{\text{PF!Requirement(name,rqref,rqref)}}{\text{RE!UC(name,relateduc,UCRelation)}} \tag{6}$$

R_6 handles the transformation of domains from Problem Frames into actors. If a domain is identified as an actor through its requirement references, it will be assigned to an actor with a name and associated use cases.

$$R_7 : \frac{\text{PF! PFModel(name, domain)}}{\text{RE! DomainModel(name, entity)}} \tag{7}$$

R_7 transforms the requirements from Problem Frames into use cases. The names of the requirements and related use case relations are mapped to the names, related use cases, and UCRelation of the use cases.

4.3.4 PF2ServiceDiagram

In the Problem Frames model, Event elements describe system-level actions or interactions triggered by requirements, while Value elements capture the data involved in those actions.In the UML Service Diagram, the central element is Service, which aggregates multiple Operation elements to represent available functionalities. Each Operation includes a name, a set of Parameters, and may reference internal Attributes for temporary data. The TypeCS element defines the type constraints for parameters and attributes.Given their semantics, Event elements map naturally to Operation, and Value elements serve dual roles: they become both Parameters of operations and Attributes of the containing service. Fig. 14 clearly presents the mapping path of Events and Values in the Problem Frames model to the UML service model.

The Problem Frames model to UML model transformation for the Use Case diagram part includes four transformation rules:

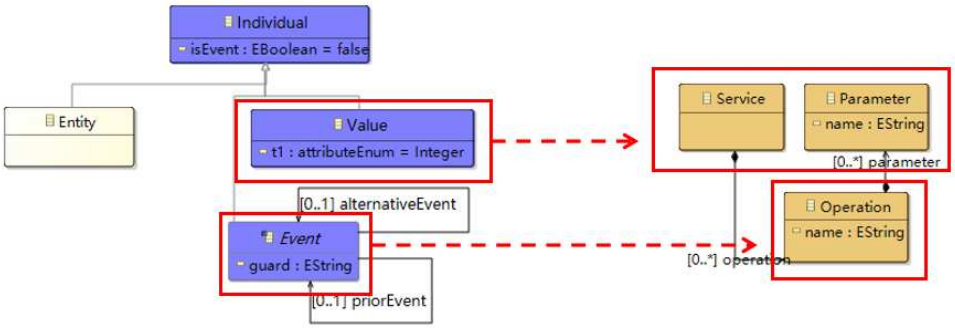


Fig. 14 Mapping of Event and Value Elements from PF to UML Service Diagram

$$R_8 : \frac{\text{PF!Requirement(name,PF!Event)}}{\text{(RE!ServiceModel(name,operation))}} \tag{8}$$

R_8 details the transformation of requirements from the PF model into the top-level service model in UML, including the name of the service and associated operations.

$$R_9 : \frac{\text{PF!Event(name,PF!Value)}}{\text{RE!Operation(name,parameter)}} \tag{9}$$

R_9 describes the process of transforming events in problem frames into operations in UML, involving the translation of event names and related values into the names and parameters of operations.

$$R_{10} : \frac{\text{PF!Value(name,type)}}{\text{(RE!Attribute(name,arrType))}} \tag{10}$$

R_{10} maps the name and type of Value to the name and type of Attribute.

$$R_{11} : \frac{\text{PF!Value(name)}}{\text{RE!Parameter(name)}} \tag{11}$$

R_{11} is responsible for mapping values from the Problem Frames as parameters of operations, encompassing only the names of the values.

4.3.5 PF2SystemSequenceDiagram

In the Problem Frames model, Phenomenon elements describe observable interactions between domains, such as signal exchanges or information flow, while State and Role elements express logical structure and participant roles. In the UML System Sequence Diagram, Interaction serves as the root structure, which aggregates Participants (Actors and Services), Messages (CallMessage, ReturnMessage), Executions, CombinedFragments, and abstract endpoints like ExecutionEnd, MessageEnd, and OperandEnd. These elements correspond directly to the behavioral elements in the problem frames: The phenomenon maps to the flow of messages and executions, the state maps to the conditional logic, and the role represents the decision points and the operand branches. This close semantic alignment allows the transformation rules to

preserve both interaction semantics and control flow, ensuring that the resulting UML diagrams accurately reflect the system behavior described in the problem model. Fig. 15 visualizes the transformation logic between the behavioral control elements in the Problem Frames model and the UML System Sequence diagram.

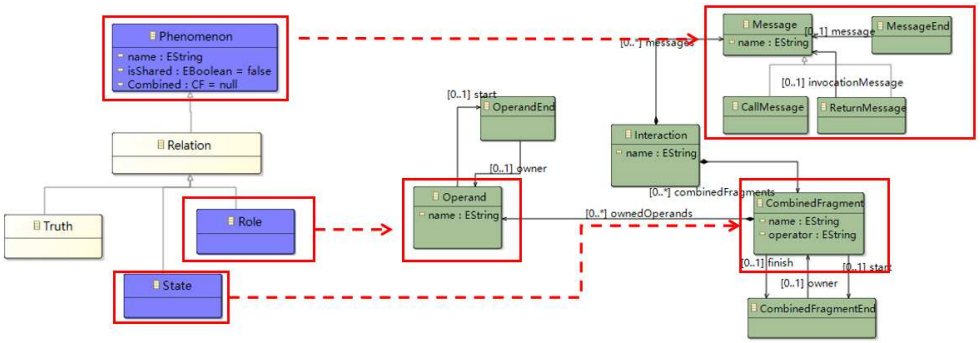


Fig. 15 Mapping of Phenomenon, State, and Role Elements from PF to UML System Sequence Diagram

The rules for transforming Problem Frames models into the SystemSequence Diagram portion of UML models are as follows:

$$R_{12}: \frac{\text{PF!Requirement(name,RE!Service} \cup \text{RE!Actor,PF!State,PF!Role,PF!Phenomenon)}}{\text{RE!Interaction(name,participants,ends,executions,messages,combinedFragments)}} \tag{12}$$

R_{12} describes the transformation of requirements from the PF model into dynamic interactions in the sequence diagram of the UML system. This rule covers the transformation from requirement names to interaction names and associates the corresponding actors (services and actors), endpoints, execution processes, messages, and combined fragments, among other properties.

$$R_{13}: \frac{\text{PF!Phenomenon(name+'StartEnd',RE!Execution,RE!Service)}}{\text{RE!ExcutionEnd(name,execution,context)}} \tag{13}$$

$$R_{14}: \frac{\text{PF!Phenomenon(name+'FinishEnd',RE!Execution,RE!Service)}}{\text{RE!ExcutionEnd(name,execution,context)}} \tag{14}$$

R_{13} and R_{14} transform phenomena in the Problem Frames model into the start and end points of execution in the UML model, associating the corresponding names, execution and contextual properties of the transformation.

$$R_{15} : \frac{\text{PF!Phenomenon(name+'SendingEnd',RE!Actor,RE!CallMessage)}}{\text{RE!MessageEnd(name,context,message)}} \tag{15}$$

$$R_{16} : \frac{\text{PF!Phenomenon(name+'ReturnReceivingEnd',RE!Actor,RE!CallMessage)}}{\text{RE!MessageEnd(name,context,message)}} \tag{16}$$

R_{15} and R_{16} transform phenomena in the problem frames model into sending and returning messages in the UML model. The rules add the suffix SendingEnd and ReturnReceivingEnd to the names of shared phenomena in the problem model to identify the sending and returning ends of messages, which are then associated with the corresponding actors (Actor) and call messages (CallMessage). This ensures an accurate representation of message sending actions in the System Sequence diagram, defining the context and body of the message.

$$R_{17} : \frac{\text{PF!State(name+'FinishEnd'))}}{\text{RE!CombinedEnd(name)}} \tag{17}$$

$$R_{18} : \frac{\text{PF!State(name+'StartEnd')}}{\text{RE!CombinedEnd(name)}} \tag{18}$$

R_{17} and R_{18} transform states in the Problem Frames model into combined end points (CombinedEnd) in the UML model. In these rules, the names of states are appended with StartEnd and FinishEnd suffixes to represent the start and end of a combined fragment, and this new name is directly mapped to the name of the combined end point. The rules simplify the transformation process from state to endpoint, ensuring that every combined fragment in the System Sequence diagram has a clear start and end marker.

$$R_{19} : \frac{\text{PF!Role(name+'End')}}{\text{RE!OperandEnd(name)}} \tag{19}$$

R_{19} describes the transformation of roles (Role) from the Problem Frames model into operand end points (OperandEnd) in the UML model. In this rule, the role names are appended with an End suffix to indicate the end of a specific operand, and this new name is directly mapped to the name of the operand end point. This transformation ensures that the end of each logical branch in the UML System Sequence diagram is clearly marked.

$$R_{20} : \frac{\text{PF!Phenomenon(name,RE!Service,RE!ExecutionEnd)}}{\text{RE!Execution(name,owner,start,end)}} \tag{20}$$

R_{20} involves mapping phenomena (Phenomenon) from the Problem Frames model to executions (Execution) in the UML model. This rule uses the name of the phenomenon directly as the name of the execution, assigns the related service (Service) as the owner of the execution, and uses the associated execution end point (ExecutionEnd) as the start and end points of the execution. This ensures that the UML System Sequence diagram accurately represents the start and end of each execution process.

$$R_{21}: \frac{\text{PF!Phenomenon(name,RE!Operation,RE!ExecutionEnd,RE!MessageEnd)}}{\text{RE!Callmessages(name,op,sendingEnd,receivingEnd)}} \tag{21}$$

R_{21} describes how to map phenomena (Phenomenon) from the Problem Frames model to call messages (CallMessage) in the UML model. In this rule, the name of the phenomenon is used as the name of the call message, the related operation (Operation) is mapped as the operation object (op) of the message call, the execution end point (ExecutionEnd) serves as the sending end (sendingEnd), and the message end point (MessageEnd) serves as the receiving end (receivingEnd). This ensures that every message call in the UML System Sequence diagram accurately reflects the interaction behavior defined in the Problem Frames.

$$R_{22}: \frac{\text{PF!Phenomenon(name,RE!Operation,RE!ExecutionEnd,RE!MessageEnd)}}{\text{RE!ReturnMessage(name,op,sendingEnd,receivingEnd)}} \tag{22}$$

R_{22} involves mapping phenomena (Phenomenon) from the Problem Frames model to return messages (ReturnMessage) in the UML model. Under this rule, the name of the phenomenon becomes the name of the return message, the related operation (Operation) is mapped as the operation object (op) of the message, the execution end point (ExecutionEnd) is used as the sending end (sendingEnd), and the message end point (MessageEnd) serves as the receiving end (receivingEnd). This ensures the completeness and accuracy of message transmission in the UML System Sequence diagram.

$$R_{23}: \frac{\text{PF!State(name+state.Combined,RE!Actor} \cup \text{RE!Service,RE!CombinedFragmentEnd)}}{\text{RE!CombinedFragment(name,coveredParticipants,finish,operator)}} \tag{23}$$

R_{23} involves mapping states (State) from the Problem Frames model to combined fragments (CombinedFragment) in the UML model. This rule names the combined fragment based on the state and its related combined type (state.Combined), and maps participants (Actor), services (Service), and the combined fragment end point (CombinedFragmentEnd) as covered participants, the finish of the fragment, and the operator, guiding how to transform state logic from the Problem Frames into combined control structures in the UML System Sequence diagram.

$$R_{24}: \frac{\text{PF!Role(name))}}{\text{RE!Operands(name)}} \tag{24}$$

R_{24} describes how to transform roles (Role) from the Problem Frames model into operands (Operands) in the UML model. In this rule, the name of the role directly becomes the name of the operand, indicating the mapping of the role to decision paths or conditional logic in the UML System Sequence diagram.

4.3.6 Transformation Algorithm

To ensure a systematic and consistent transformation from the Problem Frames model to the UML requirement model, we designed a unified transformation algorithm that integrates the rules and processes outlined in the previous sections. This algorithm reads semantically annotated elements from .xmi files and categorizes them into domain, requirement, phenomenon, state, and role models.

Initially, the algorithm loads the complete Problem Frames model and initializes a Requirement Model object to serve as the unified output container. Then it begins by transforming the domain model. Using rules R1 to R4, it identifies all domain elements marked as class-like (isClass=true) and maps them to UML Entity elements. Their associated attributes and structural relationships are transformed into Attribute and Reference elements within a DomainModel.

Next, the algorithm transforms the use-case model. It applies rules R5 to R7 to extract all domains marked as actors (isActor=true) and all requirements marked as use cases (isUseCase=true). These are respectively mapped to UML Actor and UC elements, with the associations between them derived from reference relationships in the original model.

Then, service-related components are constructed. By applying rules R8 to R11, the algorithm maps use case requirements to Service elements. Related Event elements are transformed into Operations, and Value elements are bifurcated into Parameters and internal Attributes of services, preserving both interface and internal logic structures.

Finally, the algorithm builds the system sequence model. It applies rules R12 to R24 to convert Phenomenon elements into executable flows (Executions, Messages), State elements into control structures (CombinedFragments), and Role elements into logical branches (Operands). These elements are organized under Interaction to produce a complete UML System Sequence Diagram that captures dynamic behavior and control flow.

By following this structured sequence, the transformation algorithm ensures that the Problem Frames model is comprehensively and accurately translated into a semantically rich UML requirement model. The detailed implementation of the pseudocode is provided as Algorithm 1.

Algorithm 1: Transformation algorithm from PF to UML models

Input: Problem Frames model
Output: Requirement Model

```
1  begin
2      problemframes ← retrieve(pf.xmi);
3      for pf ∈ problemframes do
4          requirementModel ← createRequirementModel(pf.name);
5          switch transformation Type do
6              case DomainModel do
7                  domains ← retrieve(pf.xmi);
8                  for d ∈ domains do
9                      if d.isClass and d.descriptionReference ≠ null then
10                         apply R1, R2;
11                         attributes ← getAttributes(d.descriptionReference);
12                         for attr ∈ attributes do
13                             apply R3;
14                         relations ← getRelations(d);
15                         for rel ∈ relations do
16                             if rel.isReference() then
17                                 apply R4;
18             case UseCaseModel do
19                 requirements ← retrieve(pf.xmi);
20                 domains ← retrieve(pf.xmi);
21                 for req ∈ requirements and d ∈ domains do
22                     if req.isUseCase ≠ null then
23                         collect(req) ∈ REQ1;
24                         apply R6;
25                     if d.isActor ≠ null then
26                         collect(d) ∈ D1;
27                         apply R7;
28                 apply R8;
29             case ServiceModel do
30                 phenomenon ← retrieve(pf.xmi);
31                 requirements ← retrieve(pf.xmi);
32                 for req ∈ requirements do
33                     if req.isUseCase() and req.service ≠ null then
34                         collect(req) ∈ REQ2;
35                         apply R9;
36                 events ← getEvents(pf.phenomenon);
37                 values ← getValues(pf.phenomenon);
38                 for req ∈ REQ2, e ∈ events, v ∈ values do
39                     if e.ascription = req then
40                         collect(e) ∈ E1;
41                         apply R10;
42                     if v.ascription = req then
43                         collect(v) ∈ V1;
44                         apply R11;
45                 for e ∈ E1, v ∈ values do
46                     if v.belong = e then
47                         collect(v) ∈ V2;
48                         apply R12;
49             case SequenceModel do
50                 phenomenon ← retrieve(pf.xmi);
51                 requirements ← retrieve(pf.xmi);
52                 state ← getStates(relation);
53                 role ← getRoles(relation);
54                 for p ∈ phenomenon, req ∈ REQ2, s ∈ states, ro ∈ roles do
55                     if p.ascription = req and p.isShared = true then
56                         collect(p) ∈ P1;
57                         apply R13, R14, R15, R16, R17, R18, R19;
58                     if s.ascription = req and s.isCombinedFragment() then
59                         collect(s) ∈ S1;
60                         apply R17, R18, R19;
61                     if ro.ascription = req then
62                         collect(ro) ∈ RO1;
63                         apply R20;
64                 for s ∈ S1, ro ∈ roles do
65                     if ro.controlled = s.controlled then
66                         collect(ro) ∈ RO2;
67                         apply R21;
68     return requirementModel
```

5 Theoretical Validation

This section will validate the feasibility of the proposed modeling platform and model transformation methods through a case study of the autonomous driving system.

5.1 Case Description

The requirements of the autonomous driving system are extracted from studies [61–63]. The system enables autonomous vehicle navigation and control on urban roads without human intervention. It adopts a distributed architecture with three components: onboard computing units for sensor data processing, environmental perception, and route planning; sensor modules (LiDAR, cameras, radar) for real-time road and obstacle monitoring; and a central control server for multivehicle coordination and real-time map/traffic updates.

5.2 Requirements Modeling

The construction of the requirements model for autonomous driving is divided into the following six steps, as illustrated in Fig. 16:

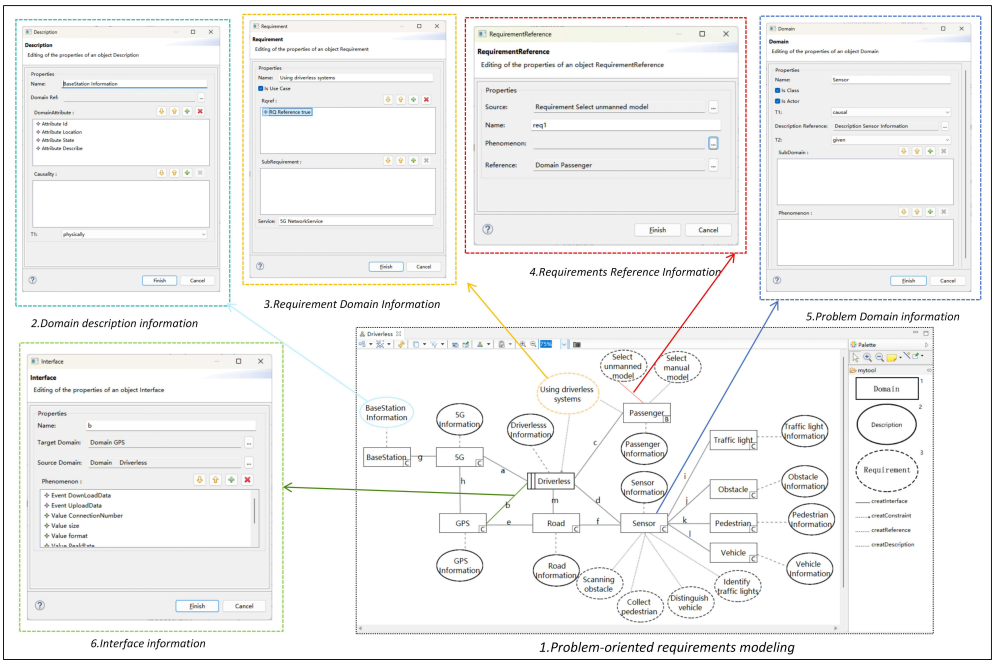


Fig. 16 Requirements Modelling for Driverless

Problem-oriented requirements model modeling: Taking the described system as the root node of the requirements model, then progressively adding other components and establishing connections according to the requirements descriptions of the system to complete the preliminary construction of the requirements model.

Domain attribute information description: Adding relevant attribute information to each domain in the requirements model to support the transformation to class diagrams.

Requirements information description: Adding information descriptions of requirements components and enhancing the effectiveness of UML use case diagrams through extended use case information descriptions.

Requirements reference/constraint information description: Adding specific dependency categories of requirements to indirectly describe whether there are implicit constraints between requirements and connected domains.

Domain ontology information description: Adding category and use case information descriptions of problem domains to improve the precision of the transformation of the requirements models.

Interface information description: Adding detailed descriptions of six types of shared phenomena for interactions between two elements connected by links to enhance the accuracy of the transformation of the UML class diagram.

5.3 Model Transformation

After the requirement model construction is completed, the PF2UML platform automatically transform it into the corresponding UML models through the above four types of transformation rules while seeking to maintain the semantic integrity of the requirements model, as illustrated in Fig. 17. This process aims to reduce semantic deviations common in manual modeling, potentially improving efficiency and helping to ensure the consistency and traceability of generated models, thereby providing support for software development.

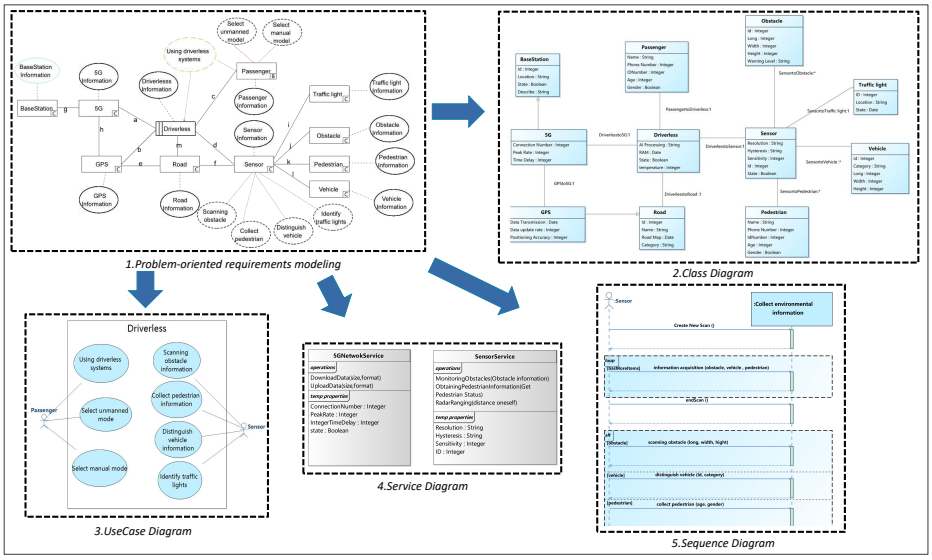


Fig. 17 Problem Frames Requirements Modelling Platform

6 Experimental Design and Analysis

This chapter aims to evaluate the efficiency and application value of the PF2UML model transformation platform by addressing three research questions.

6.1 Research Questions

To comprehensively evaluate the PF2UML transformation method, this section proposes the following three research questions:

RQ1: Study the return on investment value of the PF2UML method.

Can the return on investment (ROI) of the PF2UML method in the transformation process from extended requirements models to UML models significantly exceed the positive threshold of 100% at both the theoretical and practical levels, thus demonstrating that this method has substantial engineering value?

RQ2: Study of the improvement effect of extended meta-models on the accuracy of the PF2UML transformation.

Compared to the original PF2UML, the PF$^+$2UML model incorporates extended meta-models to enhance the semantic integrity and precision of model transformation. To clearly distinguish between the two in RQ2, we use "PF$^+$" to denote the extended problem framework metamodel and "PF" for the initial metamodel. It should be noted that, except in RQ2, "PF" elsewhere in this paper refers by default to the extended metamodel. Therefore, it is necessary to quantify the improvement in transformation accuracy brought about by PF$^+$ compared to PF in RQ2, and to identify the key value and necessity of this extension mechanism in generating high-quality UML models.

RQ3: Study on improving the efficiency of the PF2UML tools in practical application scenarios.

How can the degree of time efficiency improvement of the PF2UML modeling platform compared to traditional manual modeling methods in actual modeling tasks be quantified, and how are the practicality and promotion value of this tool in end-to-end modeling processes reflected?

6.2 Experimental Case Selection

Driven by the RQ questions, this paper developed a modeling platform prototype to validate the PF2UML research method [5] and demonstrate research feasibility. This prototype provides an intuitive way to perform problem modeling and UML model transformation, with functions including problem modeling guidance, semantic information embedding, automated model transformation, and UML model file generation. The research team recorded systematic tool demonstrations to showcase these functions. The prototype development validated the effectiveness and operability of the PF2UML method in practical applications.

To comprehensively evaluate method generalizability, this paper adopted case studies[6] covering multiple business domains: Airport Maintenance System (AirMS), Automated Teller Machine management system (ATM), Common Component Modeling Example (CoCoME), Loan Processing System (LoanPS), and Parking Management

[5] https://youtu.be/jB-G6LqmTdw
[6] https://ai4se.com/casestudy

System (ParkingPS). These cases span different industries including service, financial, and retail sectors. Through these representative instances, the experiment comprehensively evaluates the PF2UML method's adaptability and transformation stability under different complexity and requirement structure conditions.

All aforementioned cases are represented as UML models and have been thoroughly examined and approved by requirements domain experts. Therefore, we have verified reference UML models corresponding to requirements description documents, serving as standard answers for evaluating transformation accuracy.

6.3 RQ1 Experimental Design and Analysis

6.3.1 RQ1 Experimental Design

This experiment involved 12 graduate students with modeling experience who received pre-experiment training on modeling methods and platform operations to ensure consistent understanding levels. Each participant independently completed traditional requirements modeling for five cases and performed automatic conversion through the PF2UML platform to generate corresponding UML models. Participants then enhanced their traditional models with extended requirements information and performed conversion again to generate UML models for the extended requirements. The experiment was conducted in closed-book format with no communication allowed, ensuring controllable and credible results.

The PF2UML modeling platform automatically recorded the number of model inputs and generated model components, and the experimental data were used for subsequent model transformation efficiency indicator calculations.

6.3.2 RQ1 Evaluation Metrics

To scientifically assess the effectiveness and efficiency of the PF2UML method, this experiment introduces the economic term ROI (Return on Investment). ROI is calculated as a percentage, comparing investment returns with their costs to measure the impact and profitability of investments. We integrate ROI with the IEO model transformation evaluation method proposed by Professor Yang, refining it into a model transformation evaluation metric suitable for PF2UML. The refined IEO formula is expressed as follows:

$$\text{IEO} = \frac{\text{CVI}}{\text{CI}} \times 100\% = \frac{\text{EGAE - EGBE}}{\text{NES}} \times 100\% \tag{25}$$

CVI represents the return value obtained from the investment, CI represents the investment cost, EGAE denotes the total number of UML elements generated by the extended requirements model, EGBE denotes the number of UML elements directly generated by the traditional requirements model, and NES denotes the number of elements generated by the requirements model for the extended requirements model.

When IEO exceeds 100%, it indicates that our model transformation method can yield positive efficiency benefits. We will further validate the practical value of the PF2UML transformation method through calculations from both theoretical and case study perspectives in the experimental section.

6.3.3 RQ1 Experimental Results Analysis

Through dual validation of theoretical analysis and empirical research, the PF2UML method proposed in this paper demonstrates a significant return on investment value.

Theoretical level: Based on the improved IEO evaluation model, the theoretical return on investment of the PF2UML method reaches 170.8%, with system sequence diagrams achieving 325% IEO, and class diagrams and service diagrams achieving good results of 133.3% and 125% respectively, all exceeding the positive threshold of 100%(see Table 3).

Table 3 Theoretical dimensional IEO for the PF2UML method

	PF	UML	IEO (%)	Average IEO (%)
System sequence diagrams	4	13	**325**	
Class diagrams	3	4	**133.3**	**170.8%**
Use case diagram	3	3	100	
Service Diagram	4	5	**125**	

Practical level: Through empirical analysis of five typical cases (AirMS, ATM, CoCoME, LoanPS, ParkingPS), the actual average IEO value reaches 236.6%, outperforming the theoretical value. Among them, class diagrams and sequence diagrams performed most prominently, with average IEOs of 363% and 304.4% respectively, and the IEO of the ParkMS class diagrams reaching 443%(see Table 4).

Table 4 Case dimension IEO for the PF2UML method

Demo	CCD	UCD	SSD	SD	IEO
ParkMS	31/20 (155%)	12/12 (100%)	28/9 (311%)	78/29 (269%)	**281%**
LoanPS	51/31 (165%)	12/12 (100%)	84/27 (311%)	81/45 (180%)	**246%**
CoCoME	67/53 (126%)	18/18 (100%)	47/16 (294%)	119/91 (131%)	**208%**
ATM	12/9 (133%)	6/6 (100%)	98/32 (306%)	47/43 (109%)	**204%**
AirportMS	30/24 (125%)	9/9 (100%)	57/19 (300%)	33/16 (206%)	**245%**

* **CCD:** Conceptual Class Diagram; **UCD:** Use Case Diagram; **SSD:** System Sequence Diagram; **SD:** Service Diagram.

Answer to RQ1:

The experimental results show that the return on investment of the PF2UML method at both theoretical and practical levels exceeds the positive threshold of 100%, validating that this method has significant efficiency advantages and substantial engineering value in model transformation, providing an efficient transformation solution for model-driven development in the field of software engineering.

6.4 RQ2 Experimental Design and Analysis

6.4.1 RQ2 Experimental Design

For RQ2, this paper designed a comparative experiment. The experiment used five representative open-source system cases selected in Section 6.2 and their corresponding standard UML models as comparison benchmarks (Ground Truth). The experiment was divided into three phases: First, based on the configuration of Experiment 1, 12 test researchers were equally divided into two groups, using prototype tools to construct PF models and PF^+ models, respectively, and automatically generate the corresponding UML models; Second, the two types of generated UML models were compared for consistency with standard UML models; Finally, according to preset accuracy calculation rules, the transformation accuracy of the two PF models in each case was calculated.

Through this experimental design, we can not only verify the transformation accuracy of the PF2UML method, but also evaluate the improvement effect of the PF^+2UML model compared to the original PF2UML model in terms of model transformation accuracy.

6.4.2 RQ2 Evaluation Metrics

To systematically assess the accuracy improvement achieved by PF2UML after introducing extended meta-models and objectively validate the necessity of the extension mechanism, this section introduces the relative accuracy rate and the overall accuracy rate as evaluation metrics. These evaluation metrics not only reflect the transformation success rates of individual components and the overall UML models generated by this transformation method, but also help us assess the improvement effects of extended requirements models compared to the original method. The formulas are defined as follows:

$$\text{Relative Accuracy Rate} = \frac{\text{NCRE}}{\text{NSE} \cdot \mathbf{N}_p} \times 100\% \tag{26}$$

$$\text{Overall Accuracy Rate} = \frac{\text{SCS}}{\mathbf{N}_{rm}} \tag{27}$$

NCRE represents the number of elements in the generated UML model that are consistent with the standard reference model; **NSE** represents the total number of corresponding UML model elements generated; **SCS** represents the cumulative value of relative accuracy rates of various UML diagrams; $\mathbf{N_p}$ represents the total number of cases, which is 6 in this study; $\mathbf{N_{rm}}$ represents the number of types of requirements model diagrams, which is 4 in this study, namely: use case diagrams, system sequence diagrams, conceptual class diagrams, and service diagrams.

6.4.3 RQ2 Experimental Results Analysis

The comparative experiment systematically evaluated the improvement effect of extended meta-models on transformation accuracy, with results (shown in Table 5 and Fig.18) clearly demonstrating the critical value of the extension mechanism.

Table 5 Comparison of Model Transformation Results: PF2UML vs PF$^+$2UML by Case Studies

Metrics	AirportMS		ATM		CoCoME		LoanPS		ParkMS		RSR	
	PF	**PF$^+$**	**PF**	**PF$^+$**	**PF**	**PF$^+$**	**PF**	**PF$^+$**	**PF**	**PF$^+$**	**PF**	**PF$^+$**
Entity	4/8	4/4	2/11	2/2	13/16	13/13	8/14	8/8	4/8	4/4		
Reference	0/0	5/5	0/0	2/2	0/0	20/20	0/0	8/8	0/0	6/6		
Attribute	0/0	21/21	0/0	8/8	0/0	34/34	0/0	35/35	0/0	6/6		
CCD	**50.00%**	**100%**	**18.18%**	**100%**	**81.25%**	**100%**	**57.14%**	**100%**	**50.00%**	**100%**	**51.30%**	**100%**
Actor	5/8	5/5	2/11	2/2	3/16	3/3	5/14	5/5	3/8	3/3		
UC	8/8	8/8	6/6	6/6	16/16	16/16	10/10	10/10	10/11	10/10		
UCD	**81.25%**	**100%**	**47.06%**	**100%**	**59.38%**	**100%**	**62.50%**	**100%**	**68.42%**	**100%**	**63.70%**	**100%**
Operation	11/44	11/11	16/53	16/16	42/94	42/42	35/86	35/35	25/71	25/25		
Attribute	0/0	0/0	7/53	7/7	6/94	6/6	9/86	9/9	5/71	5/5		
Parameter	22/44	22/22	24/53	24/24	71/94	71/71	37/86	37/37	48/71	48/48		
SD	**37.50%**	**100%**	**29.56%**	**100%**	**42.20%**	**100%**	**31.40%**	**100%**	**36.62%**	**100%**	**35.50%**	**100%**
Execution	6/50	6/6	14/14	14/14	5/99	5/5	12/86	12/12	4/71	4/4		
Combined-Fragment	0/0	3/3	0/0	0/0	0/0	2/2	0/0	0/0	0/0	0/0		
Message	12/50	12/12	28/53	28/28	10/99	10/10	24/86	24/24	8/71	8/8		
Operand	0/0	3/3	0/0	0/0	0/0	3/3	0/0	0/0	0/0	0/0		
End	33/100	21/33	56/106	28/56	27/198	17/27	48/172	24/48	16/142	8/16		
SSD	**25.50%**	**78.95%**	**56.65%**	**71.43%**	**10.61%**	**78.72%**	**24.42%**	**71.43%**	**9.86%**	**71.43%**	**25.40%**	**74.39%**
OSR	**36.1%**	**94.74%**	**33.3%**	**92.86%**	**48.4%**	**94.68%**	**43.9%**	**92.86%**	**41.2%**	**92.86%**	**40.6%**	**93.59%**

* **CCD**: Conceptual Class Diagram; **UCD**: Use Case Diagram; **SSD**: System Sequence Diagram; **SD**: Service Diagram; **OSR**: Overall Success Rate; **RSR**: Relative Success Rate

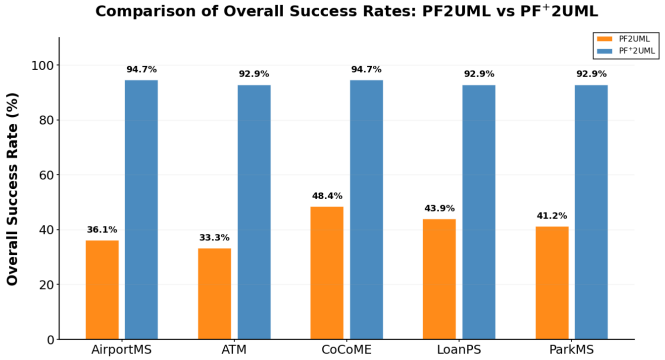


Fig. 18 Comparison of Overall Success Rates: PF2UML vs PF$^+$2UML

Overall accuracy: As shown in Table 5, the average accuracy of the extended meta-model PF$^+$2UML reaches 93. 59%, compared to 40. 6% of the original PF2UML method, with an improvement of more than 130%. Among them, the AirportMS and CoCoME cases achieved accuracy rates of 94. 74% and 94. 68%, respectively.

Accuracy of different diagram types: As shown in Table 5, at the level of various UML diagram generation, PF$^+$UML achieved 100% accuracy on conceptual class diagrams, use case diagrams and service diagrams, while the original method averaged only 51. 3%, 63. 7% and 35. 5%. The improvement in system sequence diagrams was most significant, from 25.4% of the original method to 74.39%, an improvement of nearly three times.

Answer to RQ2:

The experimental results demonstrate that the extended meta-model enhances the expressive completeness and transformation precision of generated models, with the PF$^+$2UML method achieving substantial improvements in generation accuracy across all types of UML diagrams. This extension mechanism serves as a critical factor in achieving a high-quality model transformation, providing effective technical assurance for automated model transformation.

6.5 RQ3 Experimental Design and Analysis

6.5.1 RQ3 Experimental Design

For RQ3, based on the configuration of Experiment 1, 12 recruited researchers with modeling foundations were randomly divided into two groups for comparative study. The first group used the PF2UML platform to complete the entire process from problem model modeling to automatic UML model transformation; the second group independently constructed problem models and corresponding UML models manually based on the same requirements documents without tool assistance. During the experiment, the time consumption of each participant was recorded in the two phases of problem modeling and UML construction, respectively[7].

6.5.2 RQ3 Evaluation Metrics

To quantify the degree of efficiency improvement of the PF2UML platform in actual modeling tasks, this section introduces Time Reduction as an evaluation metric to compare the differences in task completion time between platform-assisted modeling and traditional manual modeling. This metric reflects the actual benefits of the PF2UML tool in reducing modeling time and improving task execution efficiency.

For this purpose, by comparing the time consumed by two groups of experimental participants to complete the same modeling tasks, we define the time-saving rate metric as follows:

$$\text{Time Reduction} = \frac{T_{\text{Manual}} - T_{\text{PF2UML}}}{T_{\text{Manual}}} \times 100\% \tag{28}$$

TManual: represents the total time spent on completing the requirements modeling and UML model drawing manually;

TPF2UML: represents the total time spent on completing the same task using the PF2UML platform.

[7] https://www.youtube.com/playlist?list=PLl9I-9orgj5h-8IWDtcLpVUym8uoCKYVC

Based on the quantitative analysis of this metric, we can comprehensively compare the differences between the PF2UML platform and traditional manual modeling in terms of modeling efficiency and result structure performance, further verifying the applicability and value of this method in actual engineering scenarios.

6.5.3 RQ3 Experimental Results Analysis

Through a comparative analysis of the modeling time for 12 researchers in five typical cases, this article systematically evaluated the practical application efficiency of the PF2UML platform.

Table 6 Manual vs. PF2UML Efficiency Comparison Across Cases

Case Study	ATM	CoCoME	LoanPS	APMS	ParkMS	**AVG**
RM(Manual)	1740s	6900s	3480s	3180s	2460s	**3756s**
RM(PF2UML)	1200s	4200s	2100s	2460s	1620s	**2479.8s**
MT(Manual)	2070s	4770s	3300s	2835s	2772s	**3390s**
MT(PF2UML)	0.43s	3.45s	1.33s	0.13s	0.33s	**1.07s**
PF2UML(MT+RM)	1200.43s	4203.49s	2101.33s	2460.13s	1620.33s	**2480.87s**
Time reduction RM	**31.03%**	**39.13%**	**39.66%**	**22.64%**	**34.15%**	**33.98%**
Time reduction MT	99.98%	99.93%	99.96%	99.995%	99.99%	**99.97%**
Time reduction MT+RT	**42.01%**	**11.88%**	**36.32%**	**13.22%**	**41.55%**	**26.82%**

Problem diagram modeling phase efficiency: As shown in Table 6 and Fig.19, the PF2UML platform achieved an average time savings of 33. 98% in the problem diagram modeling (RM) phase, with CoCoME and LoanPS cases achieving time savings rates that exceed 39%, demonstrating the advantages of the platform in graphical modeling assistance.

Model transformation efficiency (MT): As shown in Table 6 and Fig.19, in the UML modeling phase (MT), benefitting from the automatic transformation mechanism, the platform achieved high automation with an average time savings rate of 99. 97%, with all cases approaching 100%, showing obvious advantages compared to drawing the manual UML model.

The reason for the short MT(PF2UML) transformation time (average 1.07 seconds) is that: the RM phase has already embedded relevant UML semantic information into the problem model, and the MT phase only needs to execute predefined transformation algorithms for direct mapping, avoiding the repetitive modeling process of traditional methods. To verify the overall efficiency of this "comprehensive early modeling, rapid later transformation" strategy, we designed the requirements modeling transformation efficiency indicator (RM + MT) to evaluate the comprehensive performance of the method by comparing the total time of the complete modeling process.

Comprehensive modeling efficiency (RM+MT): As shown in Table 6 and Fig.19, in the complete end-to-end modeling process, the PF2UML platform achieved an average time savings rate of 78. 64%, with the ATM case reaching the highest 81.13% and the ParkingPS case achieving 75.52%. This result validates the significant

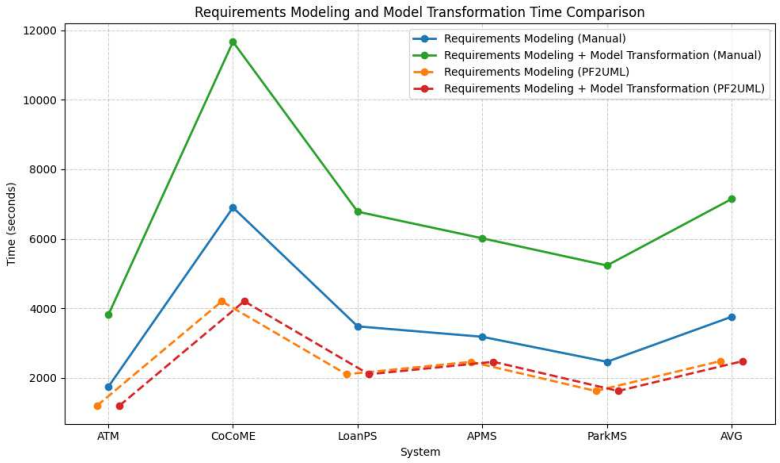


Fig. 19 Requirements Modeling and Model Transformation Time Comparison

efficiency advantages of the PF2UML method in practical engineering applications. Experimental data further indicate that although the RM phase requires additional time investment for semantic information refinement, this early investment yields substantial returns in the MT phase, significantly improving overall modeling efficiency. This result fully demonstrates the practical value of the PF2UML platform in reducing repetitive work and improving modeling automation.

Answer to RQ3:

The experimental results demonstrate that the PF2UML platform exhibits significant efficiency improvements in actual modeling tasks, with an average time saving of 78.64% in the overall modeling process. Through graphical modeling assistance and automatic transformation mechanisms, the platform effectively reduces the repetitive work of traditional manual modeling, validating the practicality and promotion value of this tool in end-to-end modeling processes.

7 Conclusion

7.1 Conclusion

This paper presents PF2UML, a model-driven approach for transforming Problem Frames models into UML models, addressing the critical gap between requirements analysis and system design. The research makes three key contributions: (1) an extended Problem Frames meta-model with semantic annotations and 24 transformation rules, (2) a graphical modeling platform based on Eclipse EMF and Sirius following MOF architecture, and (3) an automated transformation method using ATL technology. Experimental validation on five open-source systems demonstrates significant effectiveness with theoretical ROI of 170.8%, practical ROI of 236.6%, transformation accuracy of 93.59% (130% improvement), and 78.64% time reduction. The

PF2UML method successfully establishes seamless integration from requirements modeling to system design, enhancing traceability, reducing manual expertise dependence, and improving software development consistency and efficiency.

7.2 Future Work

Future research should focus on several key areas: (1) Enhanced semantic mapping through context-aware transformation rules and natural language processing integration for domain-specific requirements, (2) Tool integration with existing IDEs and CASE tools to create comprehensive development toolchains, (3) Scalability optimization including parallel processing and incremental transformation for large-scale models, (4) Comprehensive validation through longitudinal studies across diverse domains and comparative analyses with alternative approaches, (5) Extended UML support for additional diagram types such as activity and state machine diagrams, and (6) AI-enhanced transformation using machine learning for intelligent rule selection and adaptive learning mechanisms. These directions will advance the maturation and industrial adoption of model-driven requirements engineering.

Data Availability Declaration

The datasets generated and analyzed during the current study, including the PF2UML transformation platform, case study data, and experimental results, are available in the GitHub repository at https://github.com/Hongbin-Xiao/PF2UML. Additional experimental data and video demonstrations are available at the URLs referenced throughout the manuscript.

Acknowledgements

We would like to thank the graduate students who participated in our experiments for their valuable time and effort in the validation studies. We are grateful to the anonymous reviewers for their constructive comments and suggestions that helped improve the quality of this manuscript. We also acknowledge the valuable feedback received from the FSE 2025 conference, which contributed to the refinement of our research approach and methodology.

Funding

This work was supported by the National Natural Science Foundation of China (Grant No.62362006), the Guangxi Science and Technology Program (Grant No.GuiKeAB24010343), and the Innovation Project of Guangxi Graduate Education (Grant No.XYCBZ2024024).

Author Contributions Declaration

H.X. conceived the research idea, developed the PF2UML transformation methodology and 24 transformation rules, implemented the graphical modeling platform using

Eclipse EMF and Sirius framework, conducted statistical analysis, and wrote the main manuscript text. Z.L. provided research direction guidance, supervised the project, secured funding, and reviewed and polished the manuscript. B.Z. and S.Y. jointly designed and conducted experimental validation, collected and analyzed data from five open-source systems, and prepared the experimental results. Y.Y. contributed to the conceptual framework and provided expertise in model-driven engineering approaches. H.X. prepared figures 1-5 and tables 1-3. All authors reviewed the manuscript.

Competing Interests Declaration

Competing Interests Declaration

References

[1] Boehm, B.: A view of 20th and 21st century software engineering. In: Proceedings of the 28th International Conference on Software Engineering, pp. 12–29 (2006)

[2] Curcio, K., Navarro, T., Malucelli, A., Reinehr, S.: Requirements engineering: A systematic mapping study in agile software development. Journal of Systems and Software **139**, 32–50 (2018)

[3] Medvidovic, N., Taylor, R.N.: Software architecture: foundations, theory, and practice. In: Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 2, pp. 471–472 (2010)

[4] Damian, D., Chisan, J.: An empirical study of the complex relationships between requirements engineering processes and other processes that lead to payoffs in productivity, quality, and risk management. IEEE Transactions on Software Engineering **32**(7), 433–453 (2006)

[5] Rempel, P., Mäder, P.: Preventing defects: The impact of requirements traceability completeness on software quality. IEEE Transactions on Software Engineering **43**(8), 777–797 (2016)

[6] Jackson, M.: Problem Frames: Analyzing and Structuring Software Development Problems. Addison-Wesley Longman Publishing Co., Inc., Boston (2000)

[7] Rumpe, B.: Modeling with UML vol. 98. Springer, Cham (2016)

[8] Kasauli, R., Knauss, E., Horkoff, J., Liebel, G., Oliveira Neto, F.G.: Requirements engineering challenges and practices in large-scale agile system development. Journal of Systems and Software **172**, 110851 (2021)

[9] Eliasson, U., Heldal, R., Lantz, J., Berger, C.: Agile model-driven engineering in mechatronic systems-an industrial case study. In: Model-Driven Engineering

Languages and Systems: 17th International Conference, MODELS 2014, Valencia, Spain, September 28–October 3, 2014. Proceedings 17, pp. 433–449 (2014). Springer

[10] Bjarnason, E., Sharp, H.: The role of distances in requirements communication: a case study. Requirements Engineering **22**, 1–26 (2017)

[11] Liebel, G., Marko, N., Tichy, M., Leitner, A., Hansson, J.: Model-based engineering in the embedded systems domain: an industrial survey on the state-of-practice. Software & Systems Modeling **17**(1), 91–113 (2018)

[12] Hall, J., Rapanotti, L., Jackson, M.: Problem oriented software engineering: Solving the package router control problem. IEEE Transactions on Software Engineering **34**(2), 226–241 (2008)

[13] Li, X., Liu, Z., He, J.: Formal and use-case driven requirement analysis in uml. In: 25th Annual International Computer Software and Applications Conference. COMPSAC 2001, pp. 215–224 (2001). IEEE

[14] Liu, Z., Jifeng, H., Li, X., Chen, Y.: A relational model for formal object-oriented requirement analysis in uml. In: Formal Methods and Software Engineering: 5th International Conference on Formal Engineering Methods, ICFEM 2003, Singapore, November 5-7, 2003. Proceedings 5, pp. 641–664 (2003). Springer

[15] Chen, Z., Liu, Z., Ravn, A.P., Stolz, V., Zhan, N.: Refinement and verification in component-based model-driven design. Science of Computer Programming **74**(4), 168–196 (2009)

[16] Bézivin, J., Rumpe, B., Schürr, A., Tratt, L.: Model transformations in practice workshop. In: International Conference on Model Driven Engineering Languages and Systems, pp. 120–127 (2005). Springer

[17] Atkinson, C., Kuhne, T.: Model-driven development: a metamodeling foundation. IEEE software **20**(5), 36–41 (2003)

[18] Overbeek, J.: Meta object facility (mof): investigation of the state of the art. Master's thesis, University of Twente (2006)

[19] Kahani, N., Bagherzadeh, M., Cordy, J.R., Dingel, J., Varró, D.: Survey and classification of model transformation tools. Software & Systems Modeling **18**, 2361–2397 (2019)

[20] Lúcio, L., Amrani, M., Dingel, J., Lambers, L., Salay, R., Selim, G.M., Syriani, E., Wimmer, M.: Model transformation intents and their properties. Software & systems modeling **15**, 647–684 (2016)

[21] Stahl, T., Völter, M., Czarnecki, K.: Model-driven Software Development:

Technology, Engineering, Management, (2006)

[22] Kleppe, A.G., Warmer, J.B., Bast, W.: MDA Explained: the Model Driven Architecture: Practice and Promise, (2003)

[23] Budinsky, F.: Eclipse Modeling Framework: a Developer's Guide, (2004)

[24] Gronback, R.C.: Eclipse Modeling Project: a Domain-specific Language (DSL) Toolkit, (2009)

[25] Ma, Q., Kelsen, P., Glodt, C.: A generic model decomposition technique and its application to the eclipse modeling framework. Software & Systems Modeling **14**, 921–952 (2015)

[26] Madiot, F., Paganelli, M.: Eclipse sirius demonstration. P&D@ MoDELS **1554**, 9–11 (2015)

[27] López-Fernández, J.J., Garmendia, A., Guerra, E., Lara, J.: An example is worth a thousand words: Creating graphical modelling environments by example. Software & Systems Modeling **18**, 961–993 (2019)

[28] Jouault, F., Allilaire, F., Bézivin, J., Kurtev, I.: Atl: A model transformation tool. Science of computer programming **72**(1-2), 31–39 (2008)

[29] Jouault, F., Kurtev, I.: Transforming models with atl. In: International Conference on Model Driven Engineering Languages and Systems, pp. 128–138 (2005). Springer

[30] Liu, G., Li, Z., Huang, S., Ouyang, Z., Liu, Z.: Care: A computer-aided requirements engineering tool for problem-oriented software development. International Journal of Software Engineering and Knowledge Engineering **25**(09n10), 1747–1752 (2015)

[31] Chen, X., Yin, B., Jin, Z.: Dptool: A tool for supporting the problem description and projection. In: 2010 18th IEEE International Requirements Engineering Conference, pp. 401–402 (2010). IEEE

[32] Chen, X., Xiao, H., Deng, Y., Li, Z.: Nl2pd: A tool for problem diagram generation from requirements in natural language. In: 2023 IEEE 31st International Requirements Engineering Conference (RE), pp. 361–362 (2023). IEEE

[33] Zheng, B., Li, Z., Xiao, H.: A multimedia approach to problem descriptions for fine-grained detail characterization. In: 2023 IEEE 31st International Requirements Engineering Conference Workshops (REW), pp. 205–208 (2023). IEEE

[34] Wang, Y., Chen, X., Yin, L.: Timepf: A tool for modeling and verifying timing requirements based on problem frames. In: Requirements Engineering in the Big Data Era: Second Asia Pacific Symposium, APRES 2015, Wuhan, China, October

18–20, 2015, Proceedings, pp. 149–154 (2015). Springer

[35] Li, Y., Li, Z., Bu, Y., Xiao, H., Deng, Y.: Pf4md: A microservice decomposition tool combining problem frames. In: 2023 IEEE 31st International Requirements Engineering Conference (RE), pp. 359–360 (2023). IEEE

[36] Deng, Y., Li, Z., Xiao, H.: Trace4pf: A tool for automated decomposition of problem diagrams with traceability. In: SEKE, pp. 473–474 (2022)

[37] Jin, Z., Chen, X., Li, Z., Yu, Y.: Re4cps: requirements engineering for cyber-physical systems. In: 2019 IEEE 27th International Requirements Engineering Conference (RE), pp. 496–497 (2019). IEEE

[38] Hatebur, D., Heisel, M., Schmidt, H.: A formal metamodel for problem frames. In: International Conference on Model Driven Engineering Languages and Systems, pp. 68–82 (2008). Springer

[39] Colombo, P., Del Bianco, V., Lavazza, L., Coen-Porisini, A.: A methodological framework for sysml: a problem frames-based approach. In: 14th Asia-Pacific Software Engineering Conference (APSEC'07), pp. 25–32 (2007). IEEE

[40] Colombo, P., Lavazza, L., Coen-Porisini, A., Del Bianco, V.: Towards a meta-model for problem frames: conceptual issues and tool building support. In: 2009 Fourth International Conference on Software Engineering Advances, pp. 339–345 (2009). IEEE

[41] Colombo, P., DEL BIANCO, V., Lavazza, L.A., COEN PORISINI, A., *et al.*: A meta-model for problem frames: Conceptual issues and tool building support. International Journal On Advances in Software **3**, 100–113 (2010)

[42] Lavazza, L., Del Bianco Cefriel, V.: A uml-based approach for representing problem frames. In: 26th International Conference on Software Engineering-W4S Workshop" 1st International Workshop on Advances and Applications of Problem Frames (IWAAPF 2004)", pp. 39–48 (2004). IET

[43] Lavazza, L., Coen-Porisini, A., Colombo, P., Del Bianco, V.: A meta-model supporting the decomposition of problem descriptions. In: 2010 Fifth International Conference on Software Engineering Advances, pp. 50–57 (2010). IEEE

[44] Yang, Y., Cheng, Y., Gao, J., Li, Z.: Extending meta-model of problem frames for metaverse human-computer interaction system. In: 2024 IEEE 32nd International Requirements Engineering Conference Workshops (REW), pp. 184–191 (2024). IEEE

[45] Han, D., Xing, J., Yang, Q., Li, J., Zhang, X., Chen, Y.: Integrating goal models and problem frames for requirements analysis of self-adaptive cps. In: 2017 IEEE 41st Annual Computer Software and Applications Conference (COMPSAC), vol.

2, pp. 529–535 (2017). IEEE

[46] Wei, S., Li, Z., Yang, Y., Xiao, H.: Zoom4pf: A tool for refining static and dynamic domain descriptions in problem frames. In: 2021 IEEE 29th International Requirements Engineering Conference (RE), pp. 414–415 (2021). IEEE

[47] Yu, Y., Tun, T.T., Tedeschi, A., Franqueira, V.N., Nuseibeh, B.: Openargue: Supporting argumentation to evolve secure software systems. In: 2011 IEEE 19th International Requirements Engineering Conference, pp. 351–352 (2011). IEEE

[48] Yang, Y., Zeng, B., Chen, Z., Gao, J.: Pf-hcps: Extending problem frames for supporting human-cyber-physical system collaboration. In: 2023 IEEE 31st International Requirements Engineering Conference Workshops (REW), pp. 209–212 (2023). IEEE

[49] Choppy, C., Reggio, G.: A uml-based approach for problem frame oriented software development. Information and software technology **47**(14), 929–954 (2005)

[50] Lavazza, L., Del Bianco, V.: Combining problem frames and uml in the description of software requirements. In: Fundamental Approaches to Software Engineering: 9th International Conference, FASE 2006, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2006, Vienna, Austria, March 27-28, 2006. Proceedings 9, pp. 199–213 (2006). Springer

[51] Del Bianco, V., Lavazza, L.: Enhancing problem frames with scenarios and histories in uml-based software development. Expert Systems **25**(1), 28–53 (2008)

[52] Côté, I., Heisel, M., Schmidt, H., Hatebur, D.: Uml4pf—a tool for problem-oriented requirements analysis. In: 2011 IEEE 19th International Requirements Engineering Conference, pp. 349–350 (2011). IEEE

[53] Beckers, K., Beckers, K.: Supporting common criteria security analysis with problem frames. Pattern and Security Requirements: Engineering-Based Establishment of Security Standards, 195–228 (2015)

[54] Rapanotti, L., Hall, J.G., Li, Z.: Deriving specifications from requirements through problem reduction. IEE Proceedings-Software **153**(5), 183–198 (2006)

[55] Yang, S., Xiao, H., Li, Z., Xie, X.: Deriving and verifying b specifications from problem frames models via model transformation. In: 2024 IEEE 32nd International Requirements Engineering Conference Workshops (REW), pp. 206–213 (2024). IEEE

[56] Brito, I.S., Barros, J.P., Gomes, L.: Controller design and implementation: An approach based on problem frames and petri net models. In: 2015 IEEE 13th International Conference on Industrial Informatics (INDIN), pp. 860–867 (2015).

IEEE

[57] Colombo, P., Khendek, F., Lavazza, L.: Generating early design models from requirements analysis artifacts using problem frames and sysml. In: European Conference on Modelling Foundations and Applications, pp. 97–114 (2011). Springer

[58] Gao, N., Li, Z.: Generating testing codes for behavior-driven development from problem diagrams: A tool-based approach. In: 2016 IEEE 24th International Requirements Engineering Conference (RE), pp. 399–400 (2016). IEEE

[59] Kannan, K., *et al.*: An approach for decomposing requirements into analysis pattern using problem frames (drap-pf). In: 2015 International Conference on Advances in Computing, Communications and Informatics (ICACCI), pp. 2392–2396 (2015). IEEE

[60] ZhiLi, zhiJin: From user requirements to software specifications: An approach based on problem transformation. Journal of Software **24**(5), 961–976 (2013)

[61] Kugele, S., Obergfell, P., Sax, E.: Model-based resource analysis and synthesis of service-oriented automotive software architectures. Software and systems modeling **20**, 1945–1975 (2021)

[62] Lipson, H., Kurman, M.: Driverless: Intelligent Cars and the Road Ahead, (2017)

[63] De La Torre, G., Rad, P., Choo, K.-K.R.: Driverless vehicle security: Challenges and future research opportunities. Future Generation Computer Systems **108**, 1092–1111 (2020)