Forschungsberichte
der Fakultät IV – Elektrotechnik und Informatik

Model-based Semantic Conflict Analysis for Software- and Data-Integration Scenarios

Bericht-Nr. 2009 – 07

Henning Agt, Gregor Bauhoff, Jürgen Widiker,
Ralf-D. Kutsche, Nikola Milanovic

ISSN 1436-9915

Model-based Semantic Conflict Analysis for Software- and Data-integration Scenarios

Henning Agt, Gregor Bauhoff, Jürgen Widiker,
Ralf Kutsche, Nikola Milanovic
Technische Universität Berlin
Fakultät IV (Elektrotechnik und Informatik)
Fachgebiet Datenbanksysteme und Informationsmanagement
Einsteinufer. 17
D-10587 Berlin
{hagt,gbauhoff,jwidiker,rkutsche,nmilanov}@cs.tu-berlin.de
Homepage: `http://www.dima.tu-berlin.de`

March 5, 2009

Abstract

The *semantic conflict analysis*, which is the focus of this technical report, is an approach to automate various design-time verification activities which can be applied during software- or data-integration processes. Specifically, the aspects of semantic matching of business processes and the underlying IT infrastructure as well as of technical aspects of the composite heterogeneous systems will be investigated. The report is part of the BIZYCLE project, which examines applicability of model-based methods, technologies and tools to the large-scale industrial software and data integration scenarios. The semantic conflict analysis is thus part of the overall BIZYCLE conflict analysis process, comprising of semantic, structural, communication, behavior and property analysis, aiming at facilitating and improving standard integration practice. Therefore, the project framework will be briefly introduced first, followed by the detailed semantic annotation and conflict analysis descriptions, and further backed up with the semantic conflict analysis motivation/illustration scenario.

Contents

1 Introduction into BIZYCLE **4**

1.1 BIZYCLE Metamodels . . . 4

1.1.1 Computation Independent Metamodel . . . 5

1.1.2 Views of the Computation Independent Model . . . 7

1.1.3 Platform Specific Metamodels . . . 9

1.1.4 Platform Independent Metamodel . . . 9

1.1.5 Semantic Metamodel . . . 12

1.1.6 Connector Metamodel . . . 12

1.1.7 Mapping Metamodel . . . 13

1.2 Conflict Analysis within the BIZYCLE Integration Process . . . 14

2 Related work **16**

3 Motivating scenario **18**

3.1 Scenario 1: Transfer of a simple Business Object . . . 19

3.2 Scenario 1a: Mapping of Business Components to Integratable Elements . . . 20

3.3 Scenario 2: Transfer of a structured Business Object . . . 23

3.4 Scenario 2.1: Interface call order . . . 25

3.5 Scenario 2.2: Interface choice . . . 28

3.6 Scenario 2.3: Deadlock recognition . . . 30

3.7 Scenario 2.4: Interface call loop . . . 32

3.8 Scenario 3: Integrating a Domain Function . . . 35

3.9 Scenario 4: Integrating a Connector Function . . . 36

3.10 Scenario 5: Multiple Annotations . . . 38

3.11 Scenario 6: Integrating a Business Function . . . 40

3.12 Scenario 7: Simultaneous Annotation of Business Objects and Business Functions . . . 42

3.13 Summary of the Semantic Conflict Analysis Requirements . . . 45

4 Conflict Analysis Algorithm **47**

4.1 Conflict Types . . . 47

4.2 Overview of the Conflict Analysis Algorithm . . . 48

5 Semantic Annotation **51**

5.1 Data-oriented Annotations . . . 52

5.1.1 CIM Level Annotations . . . 52

5.1.2 PSM Level Annotations . . . 52

5.1.3 PIM Level Annotations . . . 54

5.2 Function-oriented Annotations . . . 55

5.3 Logical Operators . . . 57

5.4 Applications of Semantic Annotation . . . 57

5.4.1 Single Representation Annotations . . . 58

5.4.2 Multiple Representation Annotation . . . 58

5.4.3 Containment Annotation 59
5.4.4 Alternative Annotation 60
5.4.5 Compositional Annotation 60
5.4.6 Multilevel Annotation 61

6 Semantic Conflict Analysis 63
6.1 Algorithm Description 63
6.1.1 Algorithm Overview 64
6.1.2 Get Business Object Import and Export and Business Function Requirements 66
6.1.3 Create Business Object Requirement Mappings 67
6.1.4 Create Business Function Requirement Mappings 68
6.1.5 Find PIM Elements 69
6.1.6 Build PIM Element Result List 71
6.1.7 Get Additional Interface Element Requirements 72
6.1.8 Create Import Element Requirement Mappings 74
6.1.9 Semantic Reasoning 75
6.1.10 Find PIM Elements with Semantic Reasoning 76
6.1.11 Verify Business Object Requirement Mappings against Business Function Requirement Mappings 77
6.2 Conflict Analysis Results Representation 79
6.3 Conflict Analysis Results Example 81
6.3.1 Scenario 1 results: Simple requirement mappings 82
6.3.2 Scenario 2 results: Interface element context 83
6.3.3 Scenario 3 results: Connector mapping 84
6.4 Semantic Conflict Analysis Implementation 87
6.4.1 Prolog Engine 87
6.4.2 Ecore-to-Prolog Transformation 87
6.4.3 Knowledge Base & Rule Base 88

7 Summary 90

1 Introduction into BIZYCLE

This report documents part of the BIZYCLE interoperability platform, a joint industry/academy R&D effort to investigate in large-scale the potential of model-based software and data integration methodologies, tool support and practical applicability for different industrial domains [36, 37, 41]. The consortium consists of six industrial partners and academia and is part of the program of the German government, the Berlin-Brandenburg Regional Business Initiative BIZYCLE (www.bizycle.de). The long-term goal is to create a model-based tool generation and interoperability platform, in order to allow for improved and partially automated processes in software component and data integration. These integration tasks are performed by experienced software engineers and application experts, manually programming the *connectors* i.e. *the glue* among software components or (sub)systems. This requires domain-specific as well as integration requirements' analysis skills. It is a very expensive procedure, estimated between 50 and 80 per cent of the overall IT investments (see e.g., [44]). In order to reduce this cost factor, the BIZYCLE initiative develops a methodology, tools and metatools for semi-automated integration according to the MDA paradigm.

1.1 BIZYCLE Metamodels

With Model Driven Architecture (MDA) and Model Driven Development (MDD), the Object Management Group (OMG) has defined a standard process of modeling software systems by a top-down approach on three different levels of abstraction: computation independent (CIM), platform independent (PIM) and platform specific (PSM). BIZYCLE takes advantage of this approach and supports all three model levels (CIM, PIM, PSM) for the purpose of developing a process for integration of software artifacts, components, systems or data (hereinafter referred to as artifacts).

Integration with BIZYCLE starts on CIM level specifying the integration process and requirements in terms of an integration scenario. Unlike the MDA approach (CIM to PIM to PSM), the description of software artifacts themselves starts on PSM level. The artifacts are treated as black boxes and only their interfaces are taken into account. The artifacts are not initially described on PIM level because they already exist and cannot be modified. Platform specific descriptions of different artifacts are transformed to PIM level for comparison and composition conflict analysis. BIZYCLE offers a set of metamodels on each of the MDA levels. Table 1 gives an overview of the BIZYCLE metamodels.

The *Computation Independent Metamodel (CIMM)* offers elements for the description of an integration scenario with its artifacts, processes and abstract data exchanges.

Various *Platform Specific Metamodels (PSMM)* are used to describe homogeneous platform specific interface *families*, e.g., SQL database interfaces, Web Services, SAP R/3 BAPI and IDOC interfaces, XML files, J2EE and .NET components.

Level	Purpose	Metamodels	Multilevel Metamodels	
Computation independent	Business scenario	CIMM		SMM, AMM
Platform independent	Conflict analysis	PIMM	BMM, PMM	
Platform specific	Technical interfaces	PSMMs		

Table 1: BIZYCLE metamodel overview

The *Platform Independent Metamodel (PIMM)* represents the common basis for interface description of the integration artifacts. Model instances of that metamodel are created using model-to-model transformation of PSMs, currently realized with ATLAS transformation language (ATL)[1].

General properties and attributes of the integration artifacts are organized in multilevel metamodels to be able to use them on all three MDA levels. With the *Semantic Metamodel (SMM)* it is possible to build an ontology of the integration domain. The knowledge contained in the ontology is used to declare the meaning of the artifact's details on all three MDA levels. For behavioral and non-functional properties of the artifacts, such as call order and quality of service, the Behavior Metamodel (BMM) and Property Metamodel (PMM) were defined.

The multilevel metamodels are linked to the metamodels of the CIM, PIM and PSM level. Content of the semantic model can be shared between different levels. The Annotation Metamodel (AMM) realizes the linkage between all levels of abstraction.

Interface descriptions on PSM and PIM level can be divided into five categories:

- semantic annotation
- behavioral description
- (non-functional) properties
- data structure and data types
- communication properties

As the first three categories are already shared between PSM and PIM level the transformation from PSM to PIM only concerns structure and communication.

1.1.1 Computation Independent Metamodel

The computation independent metamodel (CIMM) describes an integration scenario from an abstract business perspective. The main purpose of modeling a scenario at computation independent model (CIM) level is to define integration requirements. The CIMM provides means to name artifacts that are involved in the integration, to model the process of the integration in terms of activities that shall be performed by them and transitions between them as well as structural and data exchange aspects, regardless of the technical system's details. Figure 1 shows the main CIMM metaclasses.

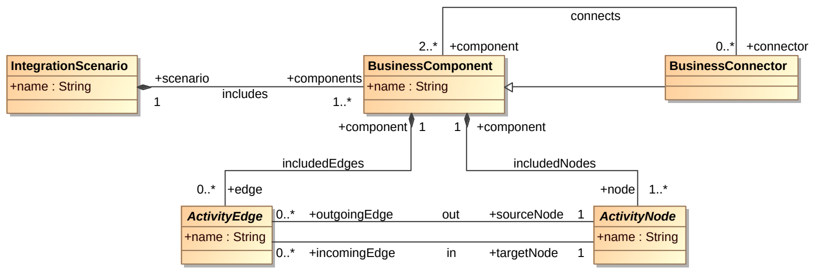


Figure 1: CIMM – Basic metaclasses for artifact and process modeling

IntegrationScenario is basically used as root model element because of implementation with Ecore, that make use of containment hierarchies in tree-based editors. A BusinessComponent represents software or data artifact, that is used in the integration. A special business component is the BusinessConnector, that stands for the interconnection between different systems. It handles the interoperability of the artifacts, that cannot be performed by themselves. ActivityNode and ActivityEdge are used to express the integration process (flow).

Figure 2 depicts different activities that can be performed by a business component. ControlNodes are used to determine the beginning and the end of an integration scenario. ExportInterface and ImportInterface are used to model component interaction by data transfer or functional coupling. Using the InternalComponentAction it is possible to express activities that are not relevant to data flow but are helpful for understanding the overall scenario context.

The metamodel also defines the metaclass BusinessFunction that represents functionality of the integrated artifacts. In the scope of an integration scenario it processes incoming data or performs a task needed by another artifact. The business function is used inside of business components while ConnectorFunction are part of business connectors. They can be used by the business architect to predefine connector functionality. The Figure 2 shows an excerpt from the abstract Enterprise Application Integration (EAI) patterns [29]. The Aggregator or Filter for instance perform data transformation tasks, Timer is used as a control function.

The CIMM defines two different kinds of edges (shown in figure 3), that can connect activities of business components. The purpose of the metaclass ControlFlow is the transition from one activity to another inside an artifact though being part of it. Connection instead models the relationship between

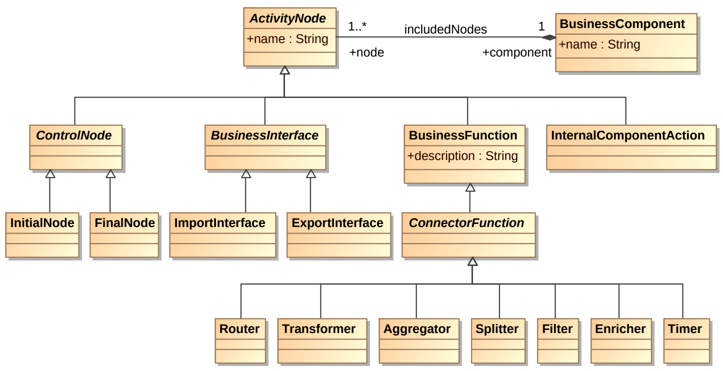


Figure 2: CIMM - Activities of BusinessComponents

different components and is responsible for the interconnection/communication tasks. Therefore it is part of a connector.

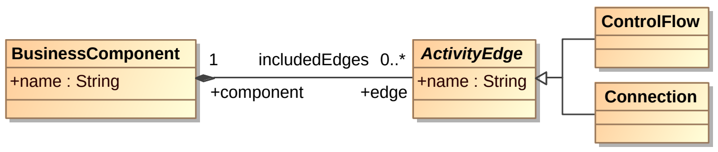


Figure 3: CIMM - ActivityEdges

Finally the CIMM defines the `BusinessObject` in order to describe data exchanged by the artifacts. Figure 4 shows the complete Computation Independent Metamodel including all relations of the business object to the other metaclasses. Business objects are produced or consumed by business functions. They are transported via connections from one artifact through an export interface to another artifact over an import interface. Furthermore it is possible to structure business objects hierarchically.

1.1.2 Views of the Computation Independent Model

The CIMM allows to model various aspects of an integration scenario, e. g., the flow and data-oriented aspects. With regard to the usability of a modeling module of the BIZYCLE model-based integration framework (MBIF), an integration scenario should not be presented on the whole to the user. We identified five potential graphical views on a CIM, which are described in the following. Table 2 gives an overview of the proposed views.

Modeling of an integration scenario starts with the *Flow View* that includes the complete process of the scenario. The visualization is similar to the UML activity diagram. It visualizes the instantiated `BusinessComponents` as swim-

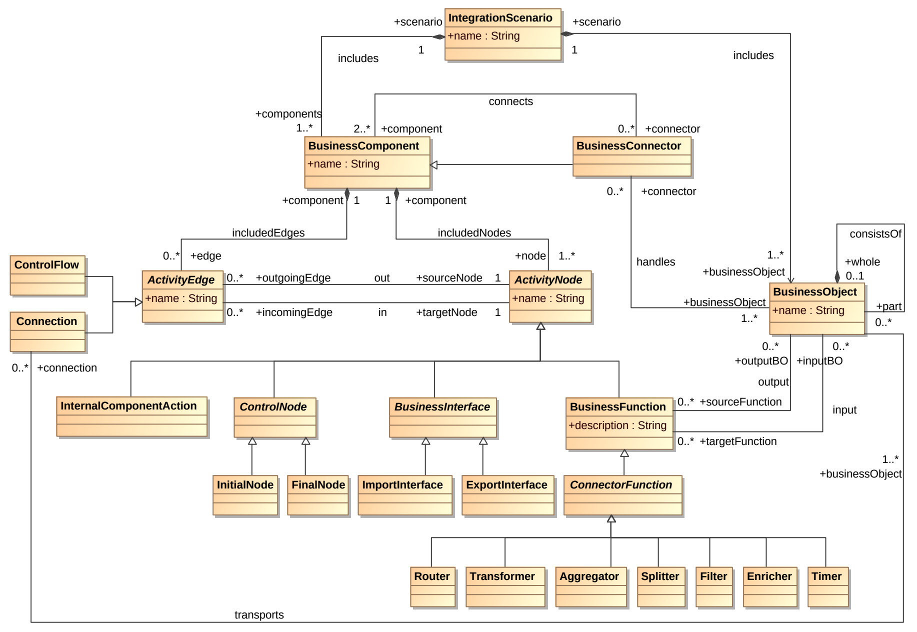


Figure 4: CIMM - Complete Metamodel

lanes, `ActivityNodes` with different shapes being part of the components and `ControlFlow` as arrows between nodes inside the swimlanes as well as `Connections` as arrows between `BusinessInterfaces` of different components.

The *Object View* is responsible for the hierarchical modeling of `BusinessObjects` refinements. Therefore only instances of this metaclass and their consistsOf-relations are shown.

The *Connection View* hides all internal activities of components and shows only data transport characteristics of the scenario. That includes `BusinessComponents` with their `BusinessInterfaces` and `Connections` and additionally the root `BusinessObjects`, that are exchanged among them. The `BusinessObjects` are associated to the `Connections` that transport them.

The *Function View* only shows `BusinessFunctions` and sub-classes including `ConnectorFunctions` as well as the `BusinessObjects`, that are input and output of the functions. The view shows the relevant business data flow describing how `BusinessObjects` are being processed and transformed into each other.

The *Semantic View* is used to semantically describe the integration scenario. Additional semantic knowledge is added in form of semantic annotations to annotatable metaclasses (`BusinessObject` and `BusinessFunction`). Both business data (objects) and functions can be semantically enriched at this level.

View	Purpose
Flow View	Flow of the integration scenario
Object View	Hierarchical structure of the business objects
Connection View	Connectivity and data transport
Function View	Business function dependencies
Semantic View	Semantic annotation of the scenario

Table 2: Computation Independent Model Views

1.1.3 Platform Specific Metamodels

At the platform specific level, BIZYCLE provides several platform specific metamodels (PSMMs) to describe technical properties of the software artifacts that are subject to the integration. With these metamodels it is possible to qualify all communicational and structural facets of the interfaces offered by each artifact. We currently support the following six types of artifacts at PSM level:

- SAP R3 JCO
- Microsoft .NET components
- J2EE EJB 2.1 components
- SQL based relational database management systems
- XML files
- Web services

Every PSM represents a description of a set of platform specific interfaces which is modeled by the application specialist. Modeling includes the semantical annotation of business objects which are pass to/from interfaces at runtime (e.g., import/export parameters). These annotations are required for the semantical Conflict Analysis at PIM level. Based on the PSMs appropriate client code can be generated which is able to call the given interfaces at runtime. This code represents the application endpoint, which wraps and hides different aspects of communication problems. Conflict analysis therefore, does not have to care about technical communication details concerning standalone interface calls. Rather, communication properties at PSM level are captured to recognize conflicts at PIM level, possibly caused due to combination of different communication protocols within the integration scenario.

More information about different PSMMs can be found in [35].

1.1.4 Platform Independent Metamodel

The purpose of the platform independent metamodel (PIMM) is to facilitate system interoperability by abstracting all platform specific heterogeneous interface details. The abstraction process is realized by a PSM-to-PIM transformation using ATL [1]. For every PSMM there exists a set of transformation rules which translates the PSM into the common abstraction layer, the PIM. As stated in [38] the PIMM facilitates integration of heterogeneous interfaces. At the PIM level it is possible to represent different interface details on a common basis.

At the PIM level an `Interface` represents a single system gateway which is able to handle data as input and/or output in one single step. Hence PIMM-Interfaces represent an abstraction for all platform specific operations, methods, functions, files etc. PIMM distinguishes between three interface types: `FunctionInterface`, `MethodInterface` and `DocumentInterface`. The first represents a non-object-oriented interface, the second an object-oriented interface and the last a data structure-based interface. Every interface has its own parent container, the system which exposes it, modeled by `IntegratableElement`. Each interface contains associations to different parts of the PIMM which will be described in the following subsections (see basic metaclasses in Figure 5).

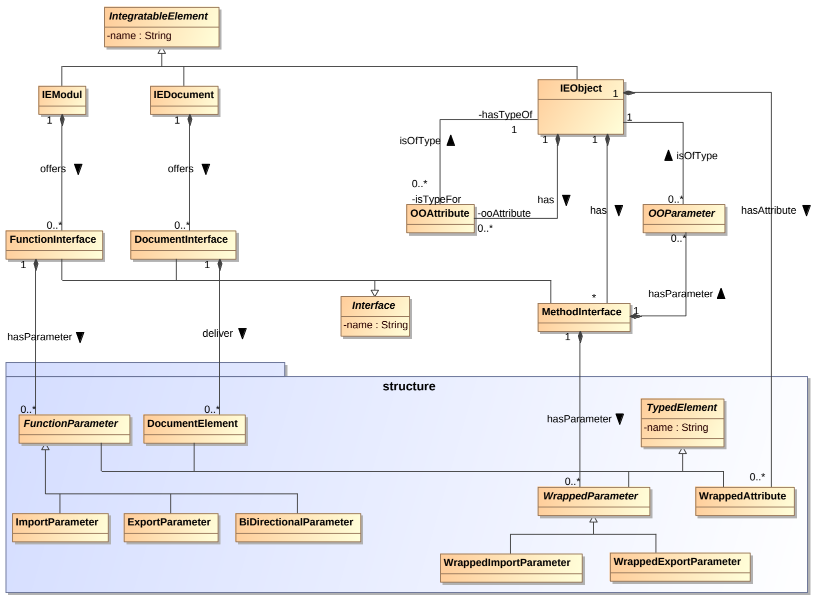


Figure 5: PIMM - basic metaclasses for interfaces (excerpt)

Structure and Communication: The structure part (Figure 6) includes the common type system with the abstract super-metaclass `Type` and different sub-metaclasses to express `SimpleTypes` (String, Number, Boolean) as well as `ComplexTypes`. The abstract `TypedElement` represents an element with a specific meaning (semantics) and represents a value container at runtime, e.g., `ImportParameter`. A TypedElement has exactly one association to a type but vice versa a type can belong to arbitrarily TypedElements, which means a Type can be 'reused'. For each interface type, abstract sub-metaclasses of TypedElement are provided: `WrappedParameter`, `FunctionParameter` and `DocumentElement`.

An interface may have its own interaction rules. These can be described by communication patterns such as `MessageExchangePatterns`, or different `Call-` and `Event` types. Knowing platform independent communication details is essential to be able to interact with every interface in the proper way. During communication conflict analysis, incompatible call mechanisms are detected and fixed (as far as possible automatically).

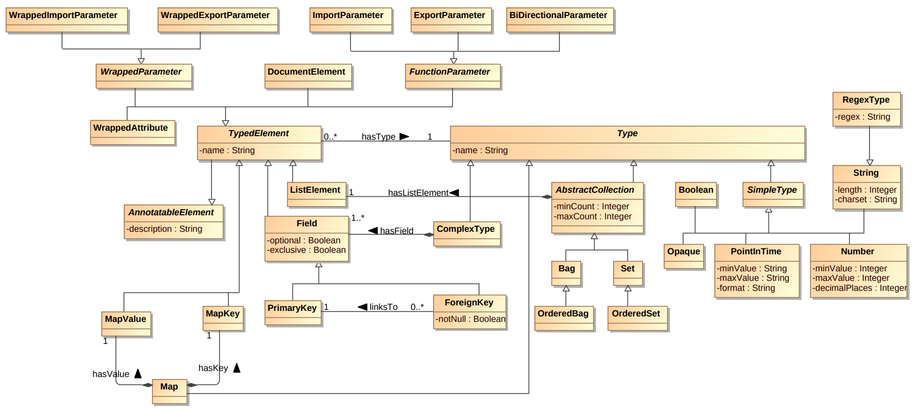


Figure 6: PIMM - structure part

Property and Behavior: Non-functional properties are used to characterize interface capability (provided) and expectations (required) properties. The root node `PropertyRoot` is responsible for including every `PropertyContainer` which is derived to `RequirementPC` and `CapabilityPC` element. Every `PropertyElement` is a child of exactly one of these two sub-metaclasses. The metaclass `QualityOfService` is the upper metaclass of every QoS property and has an association to a `Metric`. All metrics have the `AbstractMetric` as their upper metaclass. Based on this metamodel construction every QoS can be combined with every metric. During property conflict analysis, CapabilityPCs and RequirementPCs are related to each other in order to extract incompatible process flow graphs.

Every system which is represented by the metaclass `IntegratableElement` is treated as a black box, that means all implementation details are unknown. Even so, it is necessary to know at least the externalized behavior in order to follow the right communication protocol at the business level. Behavior of a component is described using OCL-inspired constraint metaclasses. There are two different possibilities to describe behavior: define interface call orders or define parameter and function constraints (conditions between parameter values, pre-conditions, post-conditions and invariants using abstract states).

1.1.5 Semantic Metamodel

The SMM defines a set of metaclasses for building semantic definitions and relations in terms of an ontology. The semantic definitions are used to declare the meaning of model elements on computation independent, platform independent and platform specific level. Having annotated the different models on these levels it is possible to compare integration artifacts more precisely and identify semantically equivalent or compatible elements. In Figure 7 the SMM metaclasses and associations are presented.

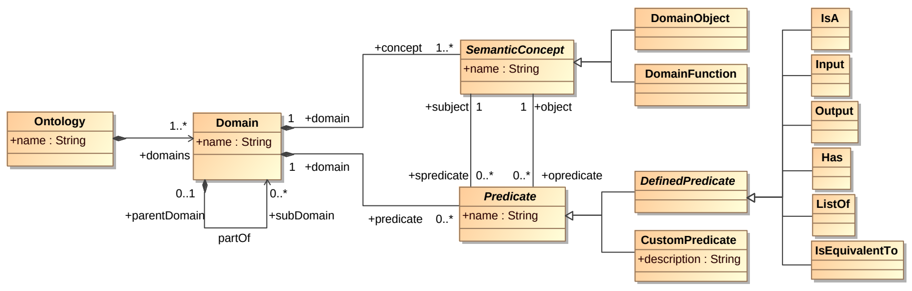


Figure 7: Semantic Metamodel

The `Ontology` can contain one or more `Domains` that define the area in which semantic descriptions should be aggredated. Ontology concepts can be `DomainObject` and `DomainFunction`. The domain object represents the knowledge about data in an integration domain, while the domain function stands for functionality representation.

Concepts are associated with `Predicates`. The domain objects and functions are either in the role of a subject or an object. With this construct it is possible to build semantic statements (RDF-like triples) consisting of subject, predicate and object. The metamodel offers a few predefined predicates such as generalization (`IsA`), data processing for functions (`Input`, `Output`), containment (`Has`), data sets(`ListOf`) and equivalence relationship (`IsEquivalentTo`). With `CustomPredicate` it is possible to model other kinds of predicates.

1.1.6 Connector Metamodel

Based on the models describing the integration scenario (at the CIM, PSM and PIM levels) and results of the conflict analysis, the connector component model and code are generated. A connector is an automatically generated component, which is used to overcome all discovered conflicts and enable technical, semantical and business interoperation. It is based on the principles of message oriented middleware (MOM), and its metamodel is accordingly based on the message passing (Figure 8).

The connector generation starts with the `ChannelAdapter` which comprises `ApplicationEndpoint` and `MessageGateway`. ApplicationEndpoint implements

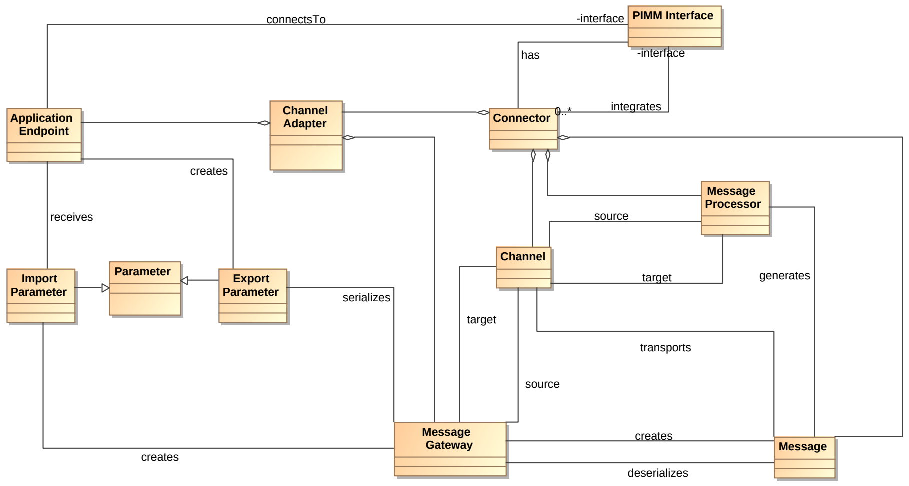


Figure 8: Connector Metamodel

the technical interoperability, and is able to call remote system interfaces. It passes export parameters to the MessageGateway, which serializes them into `Message`. In the other direction, MessageGateway deserializes a message and passes it to the ApplicationEndpoint. Messages are further transported by `Channels` which can be either 1-1 channels or publish/subscribe. `Message-Processors` perform conflict resolution by executing aggregation, routing, transformation, enrichment etc. functions. Application endpoints can be generated using standard code generation methods or using model interpretation. Core connector logic (Channels and Message Processors) are interpreted based on the UML Action Semantics description.

1.1.7 Mapping Metamodel

The Mapping Metamodel (MMM) couples the business components defined in CIM with integratable elements described in PIM. The purpose of MMM is mapping of business requirements to the real systems and components which are considered to be parts of integrated solution. Thus, the instance of MMM is dependent on the integration scenario. The mapping can be performed on two levels. One business component is mapped either to the entire integratable elements (at least one) or to the particular interfaces of integratable elements (at least one) (Figure 9). Within an integration scenario one integratable element or one interface can be mapped to multiple business components.

Since the mapping metamodel relies on the PIMM the instances of MMM are created after the PSM-to-PIM transformation and before the conflict analysis process has started. Alternatively, linking a business component the user may use MMM instances created in previous integration projects which reference the same sets of integratable elements and interfaces.

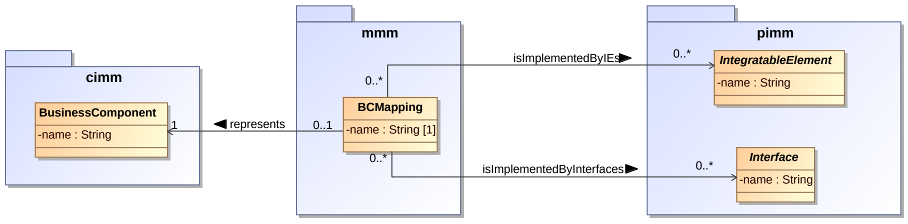
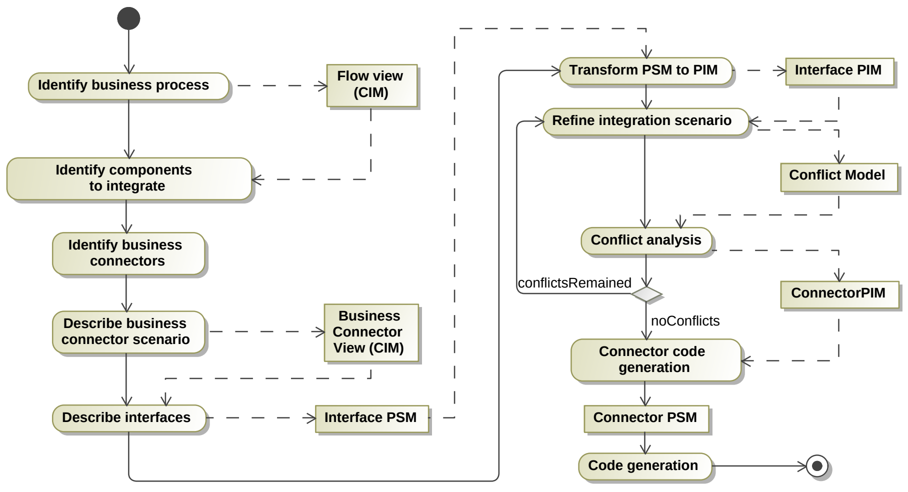


Figure 9: Mapping Metamodel

1.2 Conflict Analysis within the BIZYCLE Integration Process

Integrating heterogeneous components is burdened with recurring task, such as understanding the business semantics concerning existing technical interfaces, combining and transforming different data types, coupling conflicting functional behavior, bridging incompatible communication mechanisms, or orchestrating complex call order behavior.

Some of these tasks might be processed (semi-)automatically. Combined with a straightforward engineering process integration scenarios could be realized more efficiently. BIZYCLE differs between design-time and run-time environment. Both modules represent the *BIZYCLE interoperability platform.*



Identify business process
Flow view (CIM)
Identify components to integrate
Identify business connectors
Describe business connector scenario
Business Connector View (CIM)
Describe interfaces
Interface PSM
Transform PSM to PIM
Interface PIM
Refine integration scenario
Conflict Model
Conflict analysis
conflictsRemained
noConflicts
ConnectorPIM
Connector code generation
Connector PSM
Code generation

Figure 10: BIZYCLE integration process

The BIZYCLE integration process describes a methodology and tool-based framework which covers various integration activities, both at design time and at run time. The integration process (Figure 10) can be coarsely divided into following five steps:

1. define integration scenario at business level (CIM),
2. describe existing component interfaces at platform specific level (PSM),
3. abstract all interface descriptions to a common platform independent level (PIM),
4. process conflict analysis based on the abstracted interface descriptions and defined integration scenario including step by step conflict solution,
5. generate connector code and deploy to the BIZYCLE runtime environment.

In the remainder of this document, the conflict analysis and specifically the semantic aspects of this process, will be investigated and described in more details. We will first examine related work in the field of semantic Web and semantic reasoning, propose a motivating scenario, briefly cover the entire conflict analysis algorithm and then proceed to describe the semantic conflict analysis. The annotations will be explained and then the conflict analysis algorithm itself. The document concludes with an analysis example and prototypical implementation.

2 Related work

The term *semantic conflict* originated and still has a widespread use in the research domain of schema integration for heterogeneous database systems. It is used in the context of semantic conflict detection and corresponding resolution. The understanding of the semantic conflict analysis changed over time to include many types of compositional conflicts between any other heterogeneous data- or function-providing system.

In [42] a classification of semantic conflicts is described along the three dimensions of naming, abstraction and level of heterogeneity. It emphasizes a need of semantic reconciliation between two communication parties during a static integration or a dynamic integration approach. The order of detected semantic conflicts depends on the available subset of full schematic and semantic knowledge. Examples of semantic conflicts are: structural difference, representational difference, mismatched domains, or naming conflicts.

In [25] a formal characterization and reconstruction of the context interchange framework (COIN) strategy is given. COIN represents a mediated data access strategy in which semantic conflicts among heterogeneous systems are detected and reconciled by a context mediator through comparison of contexts associated with any two systems engaged in data exchange. Based on the logical formalism of COIN, data semantics of distinct context can be used for reasoning about semantic disparities in heterogeneous systems. Semantic conflicts occur whenever two context do not use the same interpretation of the information.

A thorough survey about ontology-based intelligent information integration is given in [49]. It analyzes about 25 approaches including including SIMS [18], TSIMMIS [21], OBSERVER, CARNOT, Infosleuth, KRAFT, PICSEL, DWQ, Ontobroker [24] , SHOE and others with respect to the role and use of ontologies. It emphasizes the need of interoperability on technical and information levels. As stated in [48] interoperability could be a key application for ontologies. They can be used for the explication of implicit and hidden knowledge to overcome the problem of semantic heterogeneity and finding suitable information sources, as part of the information retrieval and information filtering discipline. Wache et al. summarized in [49] a striking lack of sophisticated methodologies supporting development and use of ontologies. Furthermore they suggest to develop a more general methodology that includes the analysis of the integration task and support of the process of defining the role of ontologies with respect to these requirements.

Web Ontology Language (OWL)[15] is promoted as a basis for realizing different ontologies. Li and Ling [39] used it to summarize seven cases in which semantic conflicts can be encountered and resolved by an appropriate semantic conflict and resolution algorithm.

Most of the existing semantic reasoning tools rely on ontologies represented in OWL. Pellet [45] is a description logic reasoner which supports the original OWL DL specification. Similarly, Racer [26] provides support for ontology languages OWL, DAML+OIL, and RDF. The knowledge base can be queried using the query language nRQL [27]. Both reasoners implement Tableau algorithms (ABox and TBox) [19]. FaCT++ [46] is a description logic reasoner for OWL DL which augments the Tableau algorithms with optimization techniques such as preprocessing optimization, satisfiability checking optimization, and classification optimization.

OWLIM [33] is a semantic repository based on the rule-based reasoning engine TRREE [12], which supports RDF(S), OWL DLP, and OWL Horst and performs the forward chaining. Bossam [30] is a RETE-based rule engine for the Semantic Web, which enables reasoning (forward chaining) over RuleML rules, as well as ontologies described with OWL or SWRL (Semantic Web Rule Language based on combination of OWL DL with the sublanguage of RML). Hoolet [5] implements an OWL DL reasoner that uses WonderWeb OWL API [8] for parsing and managing OWL documents and Vampire [47] first oder prover as inference engine.

Jena [6] and Sesame [31] are open source RDF frameworks supporting inferencing and querying over RDF(S). Jena also provides support for OWL reasoning. Ontobroker [24] and F-OWL [50] use the frame-based approach to accomplish the semantic reasoning with ontologies represented in OWL. METEOR-S [43] is a framework for semi-automatical Web service markup with ontologies expressed in OWL. It offers several algorithms for annotation of WSDL files and their respective matching against existing ontologies. Domain ontologies are used to categorize services into domains.

The datalog-driven reasoner KAON2 [7] provides an API for managing ontologies expressed in OWL DL, SWRL [10], and F-Logic [32]. FLORA-2 [4] is a dialect of F-Logic with numerous extensions and provides its implementation based on XSB [16], a variant of the Prolog that includes HiLog [22] for higher-order programming. Reasoners are also included in various frameworks and systems as built-in components, e. g., Apollo [40] or AllegroGraph [2].

Besides XSB, there are various Prolog implementations which can be used as basis for reasoning tools depending on tool requirements (e. g., SWI-Prolog [9], BProlog [3], Visual Prolog [14], tuProlog [13], YAProlog [17]). As standard packages SWI-Prolog provides the Java Interface JPL and the Semantic Web Library for manipulating RDF documents which is used within the Prolog library Thea [11] to support the OWL documents.

3 Motivating scenario

Several motivating scenarios will be introduced in this section, with the purpose to be able to identify and formulate requirements that the semantic conflict analysis algorithm has to fulfill. The scenarios are taken from the E-commerce domain. Hence the corresponding domain ontology includes concepts such as *Customer, Address, City, TurnoverOfADay* or `BusinessFunction`s such as *NameConcatenation* (see figure 11).

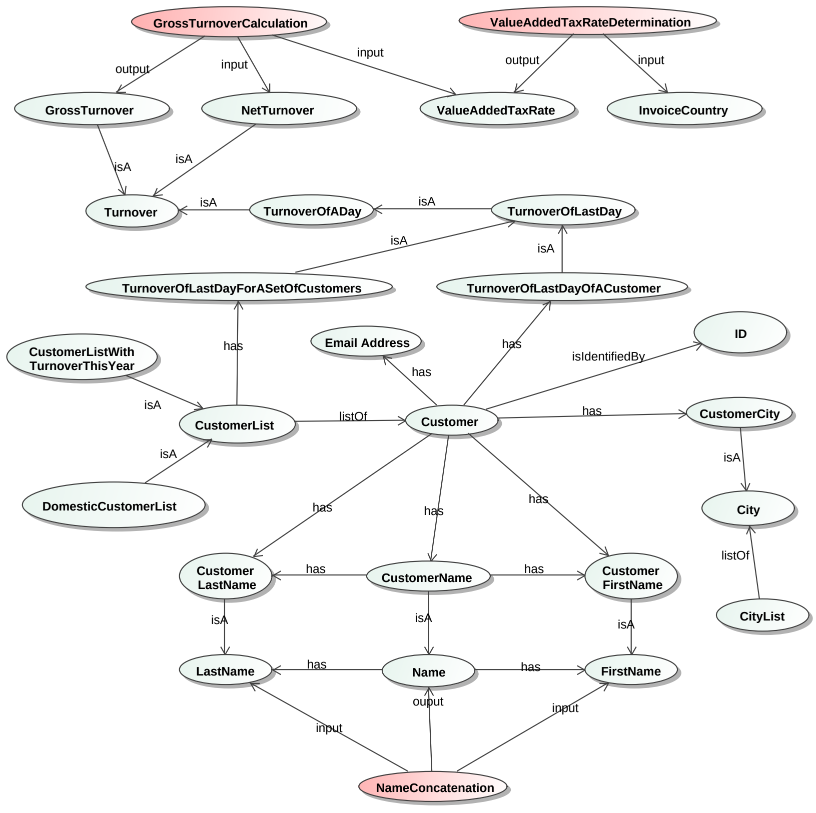


Figure 11: Ontology of the Motivating scenario

Motivating scenarios have varying complexity, where it is increased from one scenario to another. Each scenario has at least one variant without an integration conflict and further variants which cause different conflict types. The conflict analysis algorithm will be described later based on the scenarios (denoted by a number) and their variants (denoted by a letter). Every variant will be described with the expected results that the conflict analysis algorithm is expected to deliver. Scenarios are described at the platform independent (PIM) level, so terms of this level will be used.

3.1 Scenario 1: Transfer of a simple Business Object

The first scenario represents a minimal base for further discussion, consisting of two `BusinessComponents` (a Webshop and an ERP system), each with only one `FunctionalInterface`. The integration scenario, which describes the business requirements, is to transfer the sum of all sales made during the last day from the Webshop system to the ERP system. At the CIM level (see figure 12) this sum is modeled as a `BusinessObject` called `Sum of sales` and semantically annotated with the concept *TurnoverOfLastDay*.

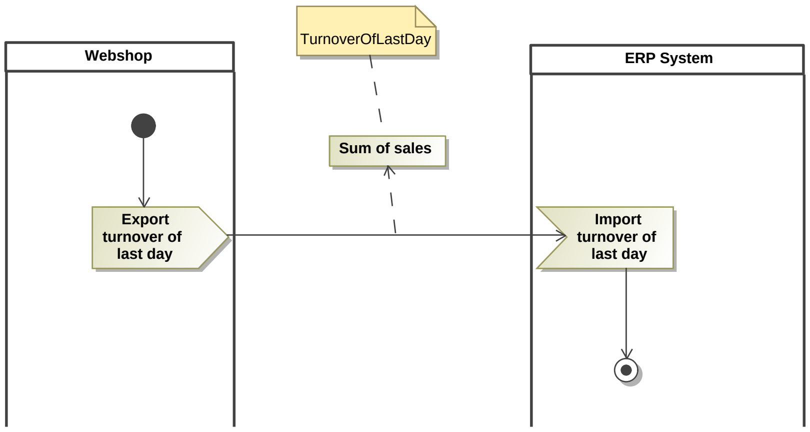


Figure 12: Scenario 1 - CIM level

At the PIM level (Figure 13), business requirements are supposed to be realized with two systems, `Webshop` and `ERP`. The Webshop has a function interface called `sumOfSalesLastDay` which delivers an export parameter `sum`, annotated with the concept *TurnoverOfLastDay*. The ERP system has another function interface `saveSumOfSalesLastDay`, which accepts an import parameter `turnover`, also annotated with the concept *TurnoverOfLastDay*.

We distinguish five possible variants of this scenario.

Variant A: All parameters and business objects are semantically annotated. The expected results of the semantic conflict analysis process are:

- Business object `Sum of sales` (CIM) is mapped to the export parameter `Webshop.sumOfSalesLastDay.sum` (PIM).
- Business object `Sum of sales` (CIM) is mapped to the import parameter `ERP.saveSumOfSalesLastDay.turnover` (PIM).

Variant B: The business object `Sum of sales` is not semantically annotated. The expected results of the semantic conflict analysis process are:

- Conflict analysis cannot map neither export parameter nor import parameter to the business object "Sum of sales", hence an integration specialist has to map it by hand (manually).

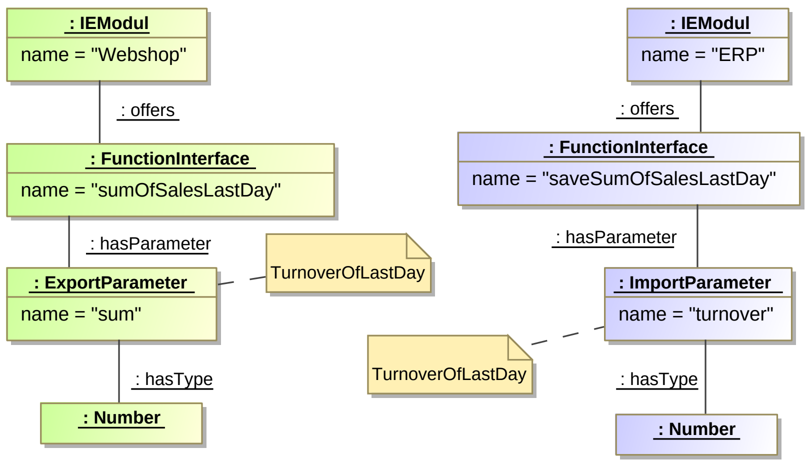


Figure 13: Scenario 1 - PIM level

Variant C: The export parameter `Webshop.sumOfSalesLastDay.sum` is not semantically annotated. The expected results of the semantic conflict analysis process are:

- Conflict analysis cannot identify an appropriate interface which delivers the defined business object. Due to this fact, an integration specialist has to map the business object `Sum of sales` to the return parameter `sum` of the functional interface `getSumOfSalesLastDay`.

Variant D: The import parameter `ERP.saveSumOfSalesLastDay.turnover` is not semantically annotated. The expected results of the semantic conflict analysis process are:

- Conflict analysis cannot identify an appropriate interface which accepts the defined business object. Equivalent to variant C, an integration specialist has to map the business object `Sum of sales` to the import parameter `turnover` of the function interface `saveSumOfSalesLastDay`.

Variants C and D can be generalized to cases where both systems don't have any functional interfaces. In those cases, they have to be manually added and annotated at the PSM level, and then transformed to the PIM level.

3.2 Scenario 1a: Mapping of Business Components to Integratable Elements

In this variation of Scenario 1, the PIM offers additional `Interface` for each `IntegratableElement`[1] at the PIM level (Figure 14). The same CIM from Scenario 1 is used (Figure 12).

[1]IEModule is a subclass of IntegratableElement

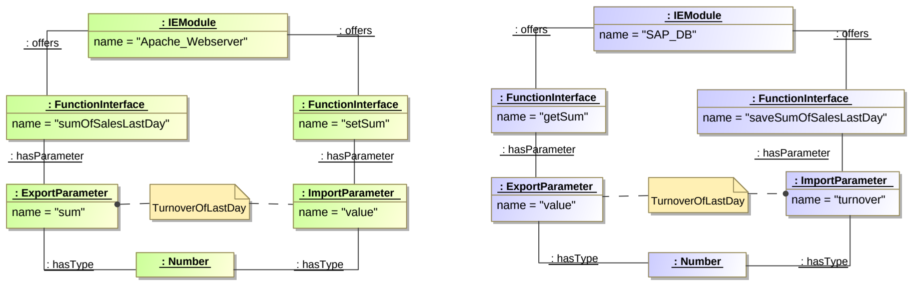


Figure 14: Scenario 1a - PIM level

This example shall emphasize the need for a manual mapping between a `BusinessComponent` at the CIM level (e.g., *Webshop*) and a set of `IntegratableElements` and/or `Interfaces` at the PIM level. The corresponding mapping metamodel is described in chapter 1.1.7 . For this scenario the *Webshop* business component is manually mapped to the `Apache_Webserver` integratable element. Similarily, the *ERP System* business component is mapped to the `SAP_DB` integratable element[2] (figure 15).

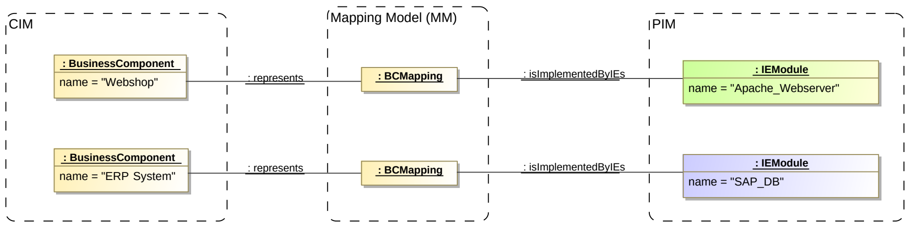


Figure 15: Scenario 1a - mapping between CIM and PIM

Variant A: Without mappings, the semantic conflict analysis will find four different tuples that satisfy the requirements defined at the CIM level. The number of possible tuples is the cardinality of the cartesian product of the sets E and I:

- E is the set of export interfaces which fulfilled the required semantics
- I is the set of import interfaces which fulfilled the required semantics

$$E \times I = \{(e, i) \mid e \in E \wedge i \in I\}$$

[2]To emphasize the difference between required (CIM) and provided systems (PIM) the `BusinessComponents` (e.g., *Webshop*) and `IntegratableElements` (e.g., `Apache_Webserver`) have different names compared to Scenario 1.

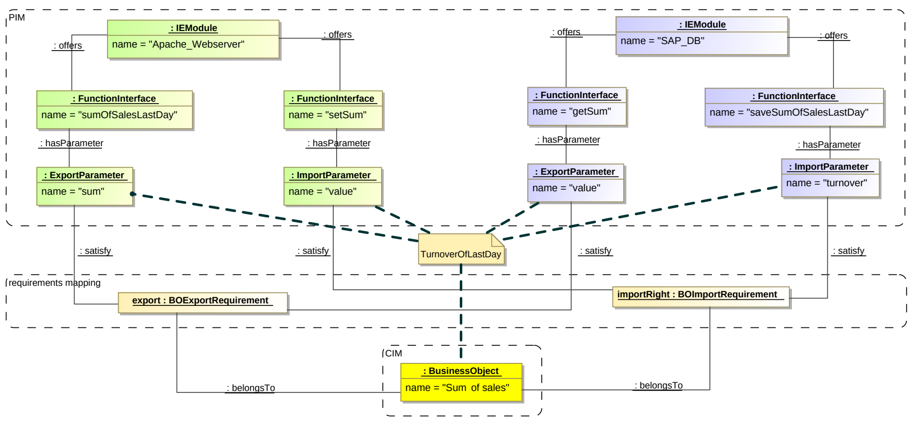


Figure 16: Scenario 1a - Result of the semantic conflict analysis without predefined mappings

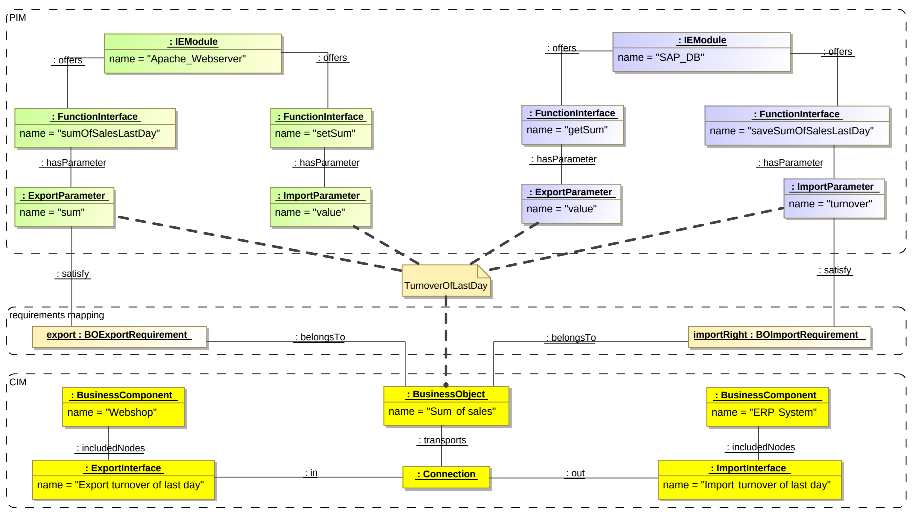


Figure 17: Scenario 1a - Result of SeCA with predefined mapping

Every tuple (e, i) represents a path which a business object can take between an export interface and an inport interface. A reflexive pathway represents an export and import on the same system. In variant A (figure 16) there are four possible paths to realize the defined integration scenario:

1. `Apache_Webserver.sumOfSalesLastDay` to `Apache_Webserver.setSum`: reflexive path

2. `Apache_Webserver.sumOfSalesLastDay` to `SAP_DB.saveSumOfSalesLastDay`:

satisfies CIM requirements

3. `SAP_DB.getSum` to `Apache_Webserver.setSum`: wrong direction

4. `SAP_DB.getSum` to `SAP_DB.saveSumOfSalesLastDay`: reflexive path

This is caused by the fact that a `BusinessComponent` at the CIM level (e.g., *Webshop*) represents only a human readable understanding of a real world system. In order to make a `BusinessComponent` machine processable (essential for the semantic conflict analysis), a precise mapping to a concrete system (represented by an `IntegratableElement`) is helpful. Without this mapping the integration specialist has to select the suitable path manually during the conflict analysis process. In complex integration scenarios the set of possible paths can become unmanageable, and predefined mappings efficiently limit the number of correct combinations, as illustrated in variant B.

Varinat B: When the mapping is used, only one path results (Figure 17): `Apache_Webserver.sumOfSalesLastDay` to `SAP_DB.saveSumOfSalesLastDay`.

3.3 Scenario 2: Transfer of a structured Business Object

The second scenario focuses on the transfer of more complex data between two systems. At the CIM level a structured business object is defined. The underlying PIM interfaces of the systems have parameters with complex types. Scenario 2 defines the base scenario for handling semantics of complex data. In scenarios 2.1 up to 2.5, the underlying PIM interfaces are modified to show different semantic aspects, that have to be covered by the conflict analysis algorithm. Figure 18 shows the visualization of the CIM model. It consists of two business components, representing a Webshop and an ERP system, as in the previous scenario. The business object that is to be transferred between them has a complex structure however. It comprises of a list of customers, each customer being identified by an ID and carrying information about his turnover for the last day. All `BusinessObjects` are annotated with concepts from the ontology described at the beginning of chapter 3. Complex business objects `ListOfCustomer` and `Customer` are annotated with concepts *CustomerList* and *Customer* respectively. Business object `CustomerID` is annotated with *ID* and `Turnover of last day` with *TurnoverOfLastDayOfACustomer*.

Both business components (Webshop and ERP) are represented at the PIM level by two modules with an interface `getCustomerListForLastDay` for the exporting side and interface `saveSumEveryCustomer` on the importing side, shown in figure 19. The interface `getCustomerListForLastDay` offers a parameter `customers` annotated with *CustomerList*, that is a list of `customer` elements annotated with *Customer*. `customerType` is a complex type consisting of three fields `id`, `name` and `turnover`, annotated with concepts *ID*, *CustomerName* and *TurnoverOfLastDayOfACustomer* respectively. The importing interface `saveSumEveryCustomer` offers the same import structure in form of a list of `cu` objects, except for the field `name` which does not exist.

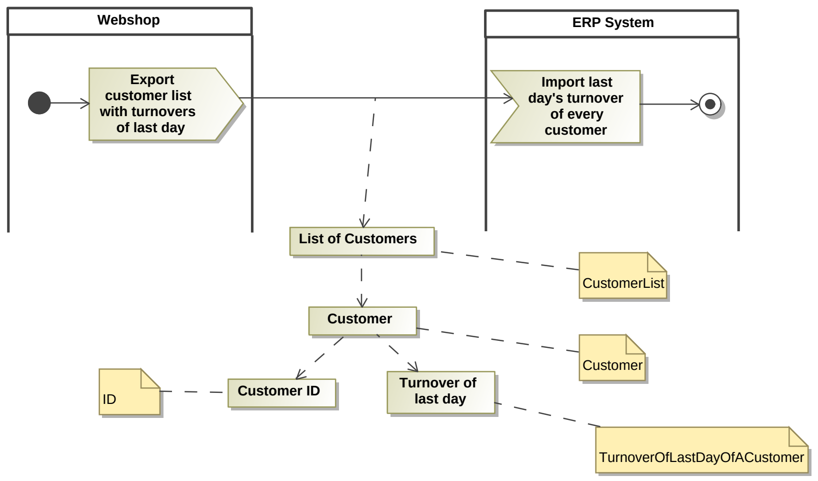

Webshop
ERP System
Export customer list with turnovers of last day
Import last day's turnover of every customer
List of Customers
CustomerList
Customer
Customer
ID
Customer ID
Turnover of last day
TurnoverOfLastDayOfACustomer

Figure 18: Scenario 2 - CIM level

The expected results of the semantic conflict analysis for this scenario are: the algorithm shall find all export and import representations at the PIM level for all business objects from the CIM level and will create the following results (the field `name` should be ignored, as it is not needed on the importing side):

- Interface `getCustomerListForLastDay` (Webshop, PIM) must be found for export
- Interface `saveSumEveryCustomer` (ERP, PIM) must be found for import
- Business object `List of Customers` (CIM) is mapped to export parameter `Webshop.getCustomerListForLastDay.customers` (PIM)
- Business object *List of Customers* (CIM) is mapped to import parameter `ERP.saveSumEveryCustomer.sumForCustomers` (PIM)
- Business object `Customer` (CIM) is mapped to Webshop export list element `customer` (PIM)
- Business object `Customer` (CIM) is mapped to ERP import list element `cu` (PIM)
- Business object `Customer ID` (CIM) is mapped to Webshop export field `customer.id` (PIM)
- Business object `Customer ID` (CIM) is mapped to ERP import field `cu.id` (PIM)
- Business object `Turnover of last day` (CIM) is mapped to Webshop export field `customer.turnover` (PIM)

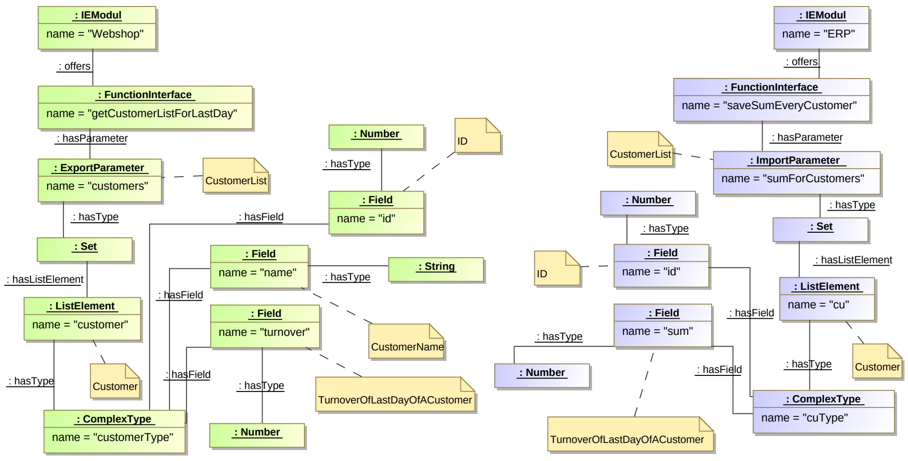


Figure 19: Scenario 2 - PIM level

- Business object `Turnover of last day` (CIM) is mapped to ERP import field `cu.sum` (PIM)
- Webshop Export field `customer.name` (PIM) is filtered out.

3.4 Scenario 2.1: Interface call order

Scenario 2.1 focuses on situations where more than one interface call is necessary for the data export. At the CIM level the business object `List of Customers` remains unchanged and still consists of customers, their IDs and turnover data as in the previous scenario. However, in this scenario the additional business object `Sum of sales`, annotated with *TurnoverOfLastDayForASetOfCustomers*, should be transfered (see figure 20).

The previous scenario is modified in such a way, that required output is generated from two Webshop interfaces, which furthermore depend on each other. Besides the original interface `getCustomerListForLastDay` the structure of the Webshop now includes an additional interface `getTotalSale`, which delivers `totalSale` as export parameter annotated with *TurnoverOFLastDayForASetOfCustomers*. It requires an import parameter `customerList` annotated with *CustomerList*. This is semantically equivalent exactly to the output of the other Webshop interface, hence two interfaces depend on each other. The structure of the ERP interface `saveSumEveryCustomer` now includes the additional import parameter `totalTurnover` annotated with *TurnoverOfLastDayForASetOfCustomers*. Figure 21 summarizes the new PIM interfaces of the Webshop and ERP.

The expected results of the semantic conflict analysis for this scenario are: the algorithm shall find all PIM export and import representations of all business objects and will create the results given below. It must further recognize

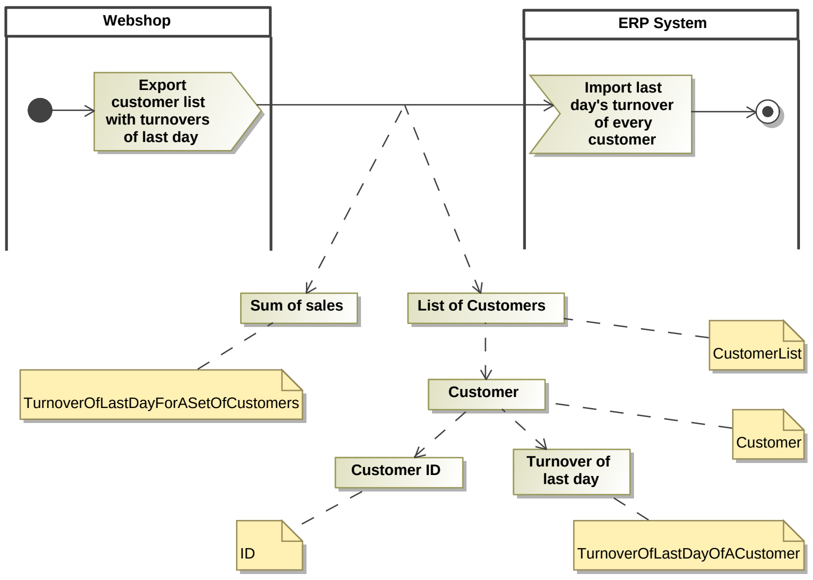

Webshop
ERP System
Export customer list with turnovers of last day
Import last day's turnover of every customer
Sum of sales
List of Customers
CustomerList
TurnoverOfLastDayForASetOfCustomers
Customer
Customer
Customer ID
Turnover of last day
ID
TurnoverOfLastDayOfACustomer

Figure 20: Scenario 2.1 - CIM level

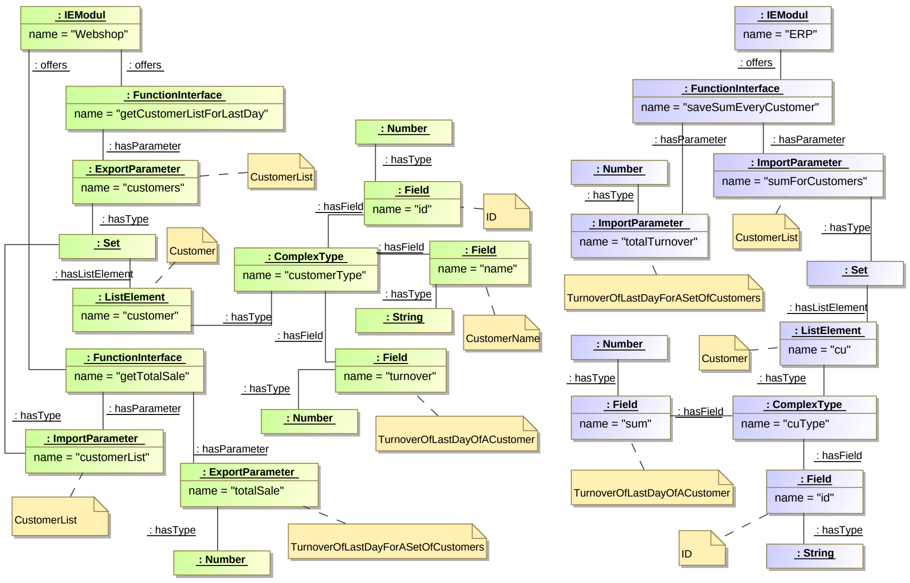


Figure 21: Scenario 2.1 - PIM level

that the import field `totalTurnover` of the ERP system can only be satisfied if the interface `getTotalSale` of the Webshop is additionally called. This interface needs the import parameter `customerList`, which can be obtained from interface `getCustomerListForLastDay`. The concrete expected output is:

- Interface `getCustomerListForLastDay` (Webshop, PIM) must be found for export
- Interface `getTotalSale` (Webshop, PIM) must be found for additional export
- Interface `getTotalSale` (Webshop, PIM) must be found for import
- The correct call order between `getCustomerListForLastDay` and `getTotalSale` must be identified, that is, `getCustomerListForLastDay` must be executed first, and its output pased to `getTotalSale`
- Interface `saveSumEveryCustomer` (ERP, PIM) must be found for import
- Business object `List of Customers` (CIM) is mapped to export parameter `Webshop.getCustomerListForLastDay.customers` (PIM)
- Business object `List of Customers` (CIM) is mapped to import parameter `ERP.saveSumEveryCustomer.sumForCustomers` (PIM)
- Business object `List of Customers` (CIM) is mapped to import parameter `Webshop.getTotalSale.customerList` (PIM)
- Business object `Customer` (CIM) is mapped to Webshop export list element `getCustomerListForLastDay.customers.customer` (PIM)
- Business object `Customer` (CIM) is mapped to ERP import list element `saveSumEveryCustomer.sumForCustomers.cu` (PIM)
- Business object `Customer` (CIM) is mapped to Webshop import list element `getTotalSale.customerList.customer` (PIM)
- Business object `Customer ID` (CIM) is mapped to Webshop export field `getCustomerListForLastDay.customers.customer.id` (PIM)
- Business object `Customer ID` (CIM) is mapped to ERP import field `saveSumEveryCustomer.sumForCustomers.cu.id` (PIM)
- Business object `Customer ID` (CIM) is mapped to Webshop import field `getTotalSale.customerList.customer.id` (PIM)
- Business object `Turnover of last day` (CIM) is mapped to Webshop export field `getCustomerListForLastDay.customers.customer.turnover` (PIM)
- Business object `Turnover of last day` (CIM) is mapped to ERP import field `saveSumEveryCustomer.sumForCustomers.cu.sum` (PIM)

- Business object `Turnover of last day` (CIM) is mapped to Webshop import field `getTotalSale.customerList.customer.turnover` (PIM)
- Business object `Sum of sales` (CIM) is mapped to export parameter `Webshop.getTotalSale.totalSale` (PIM)
- Business object `Sum of sales` (CIM) is mapped to import parameter `ERP.saveSumEveryCustomer.totalTurnover` (PIM)
- Webshop Export field `customer.name` (PIM) is filtered out.

3.5 Scenario 2.2: Interface choice

Scenario 2.2 shall demonstrate the capability of the algorithm to detect multiple possibilities for export and import of data. The requirement definition at the CIM level is the same as in the original version of scenario 2 (refer to figure 18 on page 24). The PIM interface description includes two interfaces that deliver exactly the same data and structure.

Figure 22 shows the PIM interfaces of the Webshop and ERP system. The Webshop additionally offers the interface `getCustListTurnover` with an export parameter, that has the same type as export parameter `customers` as well as the same semantic annotation. Both interfaces deliver a customer list with all relevant data, that is needed by the ERP system. Using interface `getCustList-Turnover` is an alternative to call interface `getCustomerListForLastDay`.

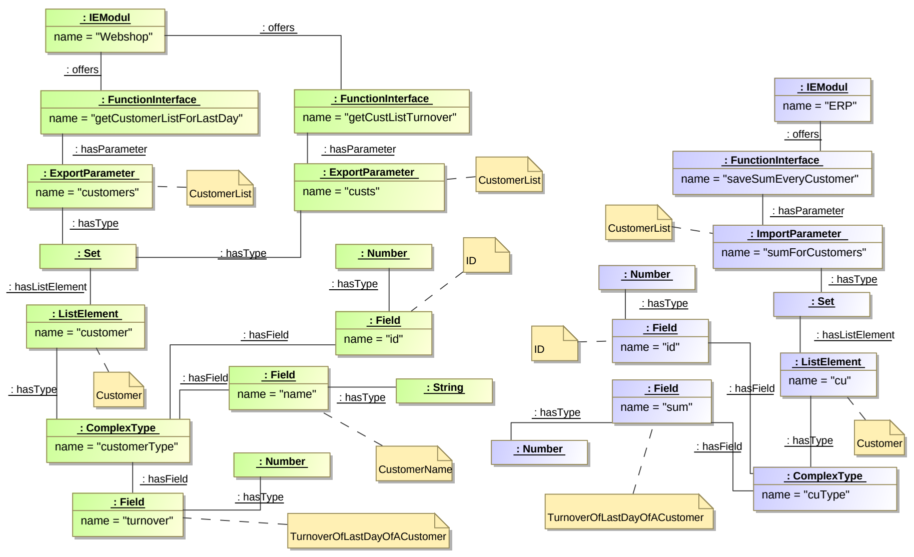


Figure 22: Scenario 2.2 - PIM level

The expected results of the semantic conflict analysis for this scenario are: the algorithm shall find all PIM export and import representations of all business objects and will create the results given below. It must recognize that it is possible to export the required data for the ERP system from interface `getCustListTurnover` as well as from `getCustomerListForLastDay`.

- Interface `getCustomerListForLastDay` (Webshop, PIM) must be found for export
- Interface `getCustListTurnover` (Webshop, PIM) must be found for export
- Interface `saveSumEveryCustomer` (ERP, PIM) must be found for import
- The alternative must be identified
- If the interface `getCustomerListForLastDay` is choosen:
  - Business object `List of Customers` (CIM) is mapped to export parameter `Webshop.getCustomerListForLastDay.customers` (PIM)
  - Business object `List of Customers` (CIM) is mapped to import parameter `ERP.saveSumEveryCustomer.sumForCustomers` (PIM)
  - Business object `Customer` (CIM) is mapped to Webhsop export list element `getCustomerListForLastDay.customers.customer` (PIM)
  - Business object `Customer` (CIM) is mapped to ERP import list element `saveSumEveryCustomer.sumForCustomers.cu` (PIM)
  - Business object `Customer ID` (CIM) is mapped to Webhsop export field `getCustomerListForLastDay.customers.customer.id` (PIM)
  - Business object `Customer ID` (CIM) is mapped to ERP import field `saveSumEveryCustomer.sumForCustomers.cu.id` (PIM)
  - Business object `Turnover of last day` (CIM) is mapped to Webshop export field `getCustomerListForLastDay.customers.customer.turnover` (PIM)
  - Business object `Turnover of last day` (CIM) is mapped to ERP import field `saveSumEveryCustomer.sumForCustomers.cu.sum` (PIM)
- If the interface `getCustListTurnover` is choosen:
  - Business object `List of Customers` (CIM) is mapped to export parameter `Webshop.getCustListTurnover.custs` (PIM)
  - Business object `List of Customers` (CIM) is mapped to import parameter `ERP.saveSumEveryCustomer.sumForCustomers` (PIM)
  - Business object `Customer` (CIM) is mapped to Webshop export list element `getCustListTurnover.custs.customer` (PIM)
  - Business object `Customer` (CIM) is mapped to ERP import list element `saveSumEveryCustomer.sumForCustomers.cu` (PIM)

  - Business object `Customer ID` (CIM) is mapped to Webshop export field `getCustListTurnover.custs.customer.id` (PIM)
  - Business object `Customer ID` (CIM) is mapped to ERP import field `saveSumEveryCustomer.sumForCustomers.cu.id` (PIM)
  - Business object `Turnover of last day` (CIM) is mapped to Webshop export field `getCustListTurnover.custs.customer.turnover` (PIM)
  - BO Business object `Turnover of last day` (CIM) is mapped to ERP import field `saveSumEveryCustomer.sumForCustomers.cu.sum` (PIM)

Actually, regardless of the types of alternative PIM objects (e.g., `customers` and `custs`) which have the equivalent semantics, the semantic conflict analysis algorithm should generate the mappings given above, because semantic annotations are matched. Eventual structural incompatibility (e.g., `custs` is chosen, but represents an ID as a string instead of expected number) will be discovered only during the structural conflict analysis. Therefore, it is important to keep track of all semantically correct combinations, as some may be invalidated during the subsequent process of conflict analysis. In that case, backtracking is required to identify alternative solutions.

3.6 Scenario 2.3: Deadlock recognition

Scenario 2.3 is based on scenario 2.1. At the CIM level the same requirements are defined (refer to figure 20 on page 26). At the PIM level all three interfaces have an additional parameter, that will cause a deadlock. The PIM interfaces are shown in figure 23. Interface `saveSumEveryCustomer` expects an import parameter `cities`, which is a list of cities. This kind of list is exported in interface `getTotalSale` through parameter `cityList`, but it is also needed as import parameter `cities` in interface `getCustomerListForLastDay`.

The expected results of the semantic conflict analysis for this scenario are: the algorithm shall find all PIM export and import representations of all business objects and will create the results below. It must recognize that calling interface `getCustomerListForLastDay` and `getTotalSale` of the Webshop will result in a deadlock, because each of the interfaces expects data from the opposite interface. It is not possible to realize this integration scenario conflict-free, unless the interfaces are modified or another interface is available that delivers additional required data.

- Interface `getCustomerListForLastDay` (Webshop, PIM) must be found for export
- Interface `getCustomerListForLastDay` (Webshop, PIM) must be found for import

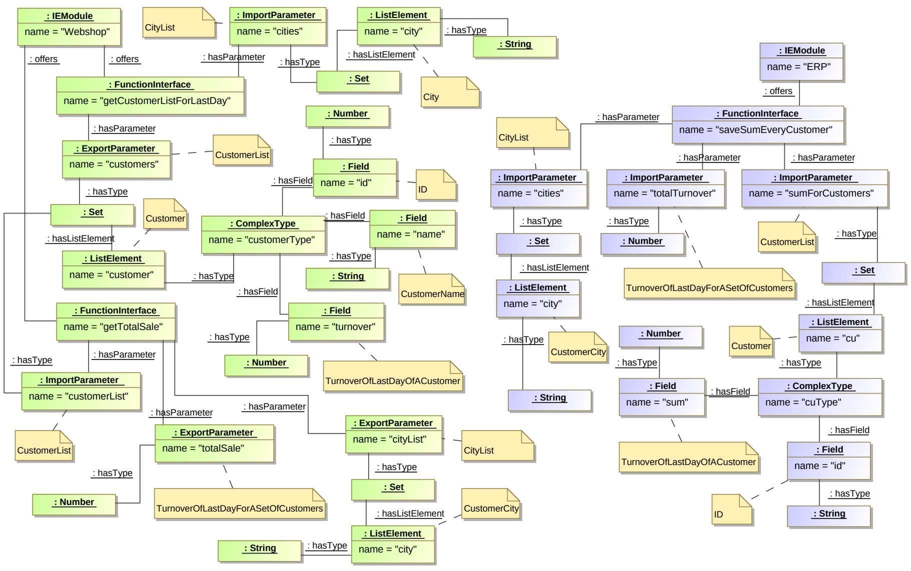


Figure 23: Scenario 2.3 - PIM level

- Interface `getTotalSale` (Webshop, PIM) must be found for export
- Interface `getTotalSale` (Webshop, PIM) must be found for import
- Interface `saveSumEveryCustomer` (ERP, PIM) must be found for import
- Business object `List of Customers` (CIM) is mapped to export parameter `Webshop.getCustomerListForLastDay.customers` (PIM)
- Business object `List of Customers` (CIM) is mapped to import parameter `ERP.saveSumEveryCustomer.sumForCustomers` (PIM)
- Business object `List of Customers` (CIM) is mapped to import parameter `Webshop.getTotalSale.customerList` (PIM)
- Business object `Customer` (CIM) is mapped to Webshop export list element `getCustomerListForLastDay.customers.customer` (PIM)
- Business object `Customer` (CIM) is mapped to ERP import list element `saveSumEveryCustomer.sumForCustomers.cu` (PIM)
- Business object `Customer` (CIM) is mapped to Webshop import list element `getTotalSale.customerList.customer` (PIM)
- Business object `Customer ID` (CIM) is mapped to Webshop export field `getCustomerListForLastDay.customers.customer.id` (PIM)

- Business object `Customer ID` (CIM) is mapped to ERP import field `save-SumEveryCustomer.sumForCustomers.cu.id` (PIM)
- Business object `Customer ID` (CIM) is mapped to Webshop import field `getTotalSale.customerList.customer.id` (PIM)
- Business object `Turnover of last day` (CIM) is mapped to Webshop export field `getCustomerListForLastDay.customers.customer.turnover` (PIM)
- Business object `Turnover of last day` (CIM) is mapped to ERP import field `saveSumEveryCustomer.sumForCustomers.cu.sum` (PIM)
- Business object `Turnover of last day` (CIM) is mapped to Webshop import field `getTotalSale.customerList.customer.turnover` (PIM)
- Business object `Sum of sales` (CIM) is mapped to export parameter `Web-shop.getTotalSale.totalSale` (PIM)
- Business object `Sum of sales` (CIM) is mapped to import parameter `ERP.saveSumEveryCustomer.totalTurnover` (PIM)
- Export parameter `Webshop.getTotalSale.cityList` (PIM) is mapped to import parameter `Webshop.getCustomerListForLastDay.cities` (PIM)
- Webshop export list element `getTotalSale.cityList.city` (PIM) is mapped to Webshop import parameter `getCustomerListForLastDay.cities.city` (PIM), based on semantic reasoning (*CustomerCity* is a *City*, see ontology in Figure 11).
- Deadlock between interfaces `Webshop.getCustomerListForLastDay` and `Webshop.getTotalSale` must be recognized, based on mappings between `getCustomerListForLastDay.cities` and `getTotalSale.cityList`, as well as between `getCustomerListForLastDay.customers` and `getTotal-Sale.customerList`

3.7 Scenario 2.4: Interface call loop

Scenario 2.4 shall demonstrate requirement for the algorithm to detect a need for consecutive interface calls. This kind of transfer occurs when a system exports list-based data (collections) and an importing system accepts single data objects only. Requirements of scenario 2.4 are the same as in scenario 2. The structured business object `List of Customers`, annotated with *CustomerList* shall be transfered, which is a collection of `Customer` objects annotated with *Customer*, consisting of `Customer ID` and `Turnover of last day`, annotated with *ID* and *TurnoverOfLastDayOfACustomer* . The underlying PIM interface of the ERP system will no longer accept a list of customers, but single customer data objects. For better readability we have included again the CIM diagram (figure 24) of section 3.3.

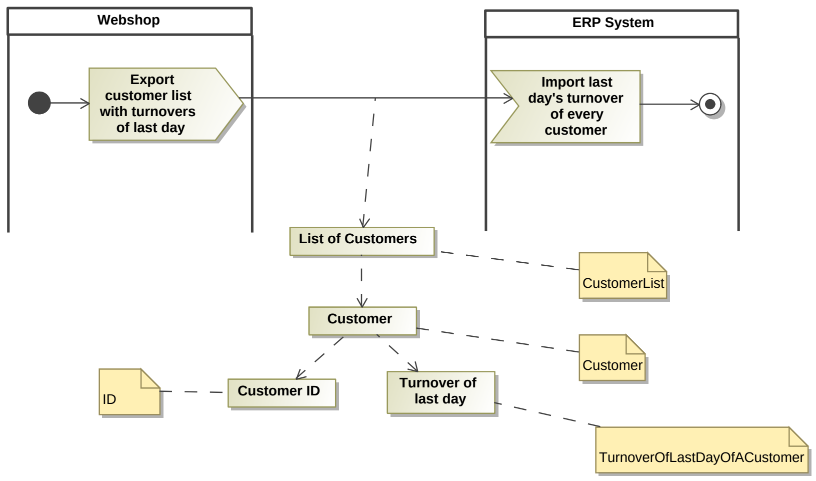

Webshop
ERP System
Export customer list with turnovers of last day
Import last day's turnover of every customer
List of Customers
CustomerList
Customer
Customer
Customer ID
ID
Turnover of last day
TurnoverOfLastDayOfACustomer

Figure 24: Scenario 2.4 - CIM level

At the PIM level (Figure 25) the Webshop is described by an interface `getCustomerListForLastDay` that offers an export parameter `customers` that is a list of customers, annotated with *CustomerList*. Elements of the list are of type `Customer` annotated with *Customer*, comprising of the fields `id`, `turnover` and `name`, annotated with *ID*, *TurnoverOfLastDayOfACustomer* and *CustomerName* respectively. The ERP system, in which the data shall be imported, offers the interface `saveSum` with two input parameters `id` and `value`, annotated with *ID* and *TurnoverOfLastDayOfACustomer*.

The expected results of the semantic conflict analysis for this scenario are: the algorithm shall find all PIM export and import representations of all business objects and will create the results below. It must be able to recognize that the elements of the list `customer` are semantically equivalent to the import parameters of the ERP system and that can be imported iteratively, by consecutive interface calls.

- Interface `getCustomerListForLastDay` (Webshop, PIM) must be found for export
- Interface `saveSum` (ERP, PIM) must be found for import
- Business object `List of Customers` (CIM) is mapped to export parameter `Webshop.getCustomerListForLastDay.customers` (PIM)
- Import mapping for the business object `List of Customers` cannot be found – no equivalent PIM parameter exists
- Business object `Customer` (CIM) is mapped to Webshop export list element `getCustomerListForLastDay.customers.customer` (PIM)

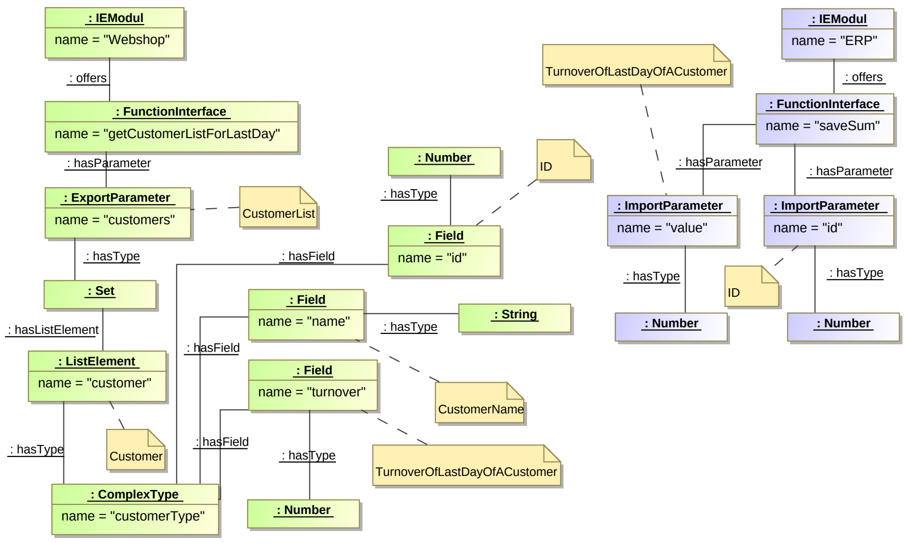


Figure 25: Scenario 2.4 - PIM level

- Import mapping for the business object `Customer` cannot be found – no equivalent PIM parameter exists
- Business object `Customer ID` (CIM) is mapped to Webshop export field `getCustomerListForLastDay.customers.customer.id` (PIM)
- Business object `Customer ID` (CIM) is mapped to import parameter `ERP .saveSum.id` (PIM)
- Business object `Turnover of last day` (CIM) is mapped to Webshop export field `getCustomerListForLastDay.customers.customer.turnover` (PIM)
- Business object `Turnover of last day` (CIM) is mapped to import parameter `ERP.saveSum.value` (PIM)
- Webshop export field `getCustomerListForLastDay.customers.customer .name` (PIM) is filtered out
- The missing import-mappings for the business objects `List of Customers` and `Customer` have to be identified as a list iteration operation, involving multiple calls of `saveSum`, that is, once for each list element from `customers`. The reasoning must be done using semantical equivalence of fields `id` and `turnover` at the Webshop side and import parameters `id` and `value` at the ERP side, as well as based on the semantic meaning of the list, being in this case a collection of identical elements.

3.8 Scenario 3: Integrating a Domain Function

In this scenario the name of the most valuable customer has to be transfered from the Webshop to the ERP system (see Figure 26). At the CIM level this transfer is modeled as the `Name` business object annotated with *CustomerName*. At the PIM level, the Webshop exports customer name as two parameters, `firstname` and `lastname` annotated with *CustomerFirstName* and *CustomerLastName*, but the ERP System accepts only one import parameter, `name` annotated with *CustomerName*. With the help of a domain function `nameConcatenation`, it is possible to concatenate first name and last name to name and thus fulfill the scenario requirements.

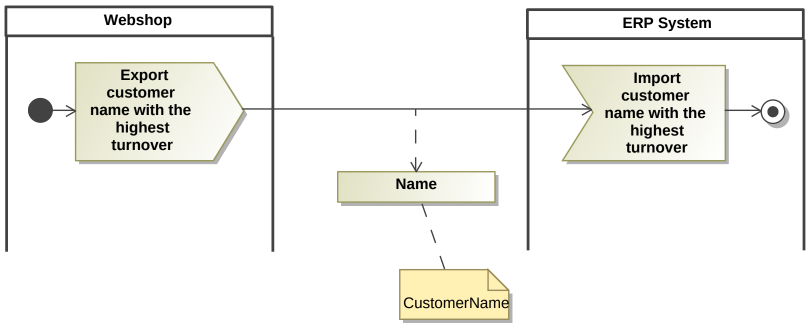

Webshop
ERP System
Export customer name with the highest turnover
Import customer name with the highest turnover
Name
CustomerName

Figure 26: Scenario 3 - CIM level

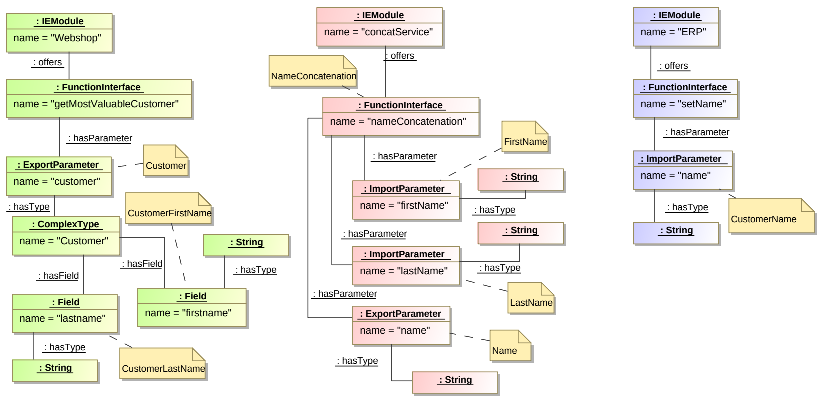


Figure 27: Scenario 3 - PIM level

The expected results of the conflict analysis are:

- An export parameter annotated with *CustomerName* cannot be found, hence, export mapping for the business object `Name` cannot be identified.

- The algorithm tries to find a domain function which delivers a value annotated with *CustomerName* as its output. The function `nameConcatenation` delivers the export parameter `name` annotated with *Name*. As the concept *Name* is an upper concept of the required *CustomerName* (connected through a "isA" predicate), business object `Name` (CIM) is mapped to export parameter `concatenationService.nameConcatenation.name` (PIM).
- Export parameter `Webshop.getMostValuableCustomer.customer.firstname` (PIM) is mapped to import parameter `concatService.nameConcatenation.firstName` (PIM) because *FirstName* isA *CustomerFirstName*
- Export parameter `Webshop.getMostValuableCustomer.customer.lastname` (PIM) is mapped to import parameter `concatService.nameConcatenation.lastName` (PIM) because *LastName* isA *CustomerLastName*
- Business object `Name` (CIM) is mapped to import parameter `ERP.setName.name` (PIM)

3.9 Scenario 4: Integrating a Connector Function

The fourth scenario includes the connector function `Add` which models a business requirement defined at the CIM level (Figure 28). Two Webshop systems export total daily turnovers as business objects `Turnover Web Shop 1` and `Turnover Web Shop 2`, both annotated with *TurnoverOfLastDay*. The connector function receives both business objects and aggregates them into the single object `Total Turnover`, also annotated with *TurnoverOfLastDay*. The ERP system receives this object.

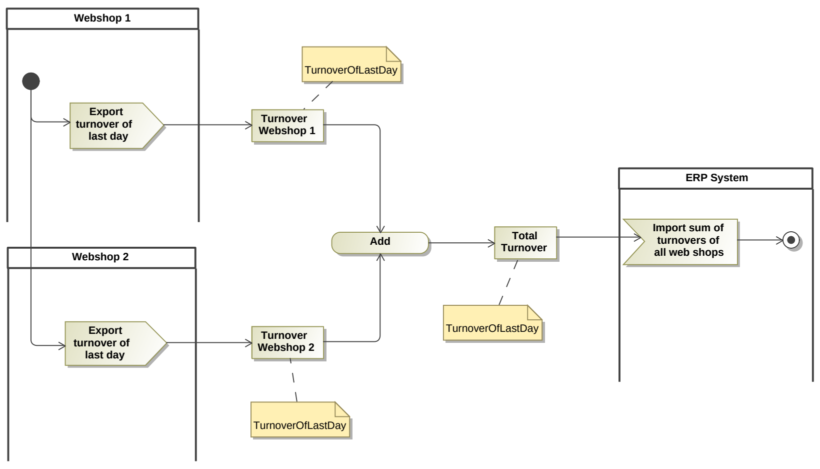


Figure 28: Scenario 4 - CIM level

At the PIM level (Figure 29, two Webshop systems have identical interfaces `getSales` which deliver export parameters `sales` annotated with *TurnoverOfLastDayForASetOfCustomers*. The ERP system has a single interface `setSales` that accept the import parameter `sales`. Export parameters from two Webshops have to be aggregated. The precise arithmetical statements describing behavior of the connector function can either be defined in an imperative, platform independent programming language like Java, or using the UML Action Semantics models, as proposed in [28].

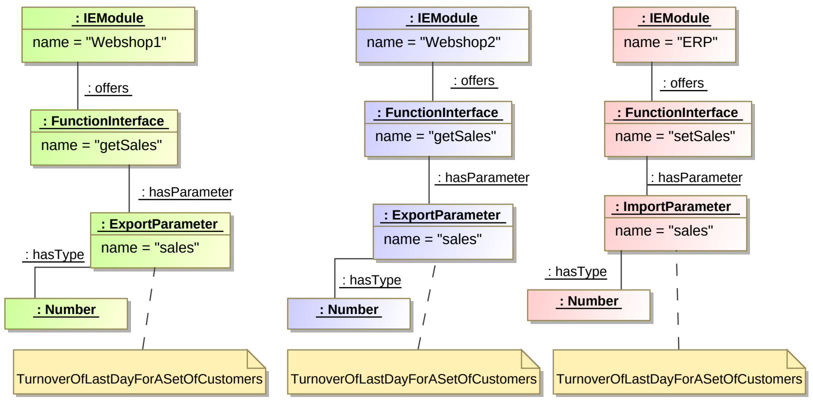


Figure 29: Scenario 4 - PIM level

The results of the semantic conflict analysis are:

- Buisness object `Turnover Webshop 1` (CIM) is mapped to export parameter `Webshop1.getSales.sales` (PIM) because *TurnoverOfLastDay* is an upper concept of *TurnoverOfLastDayForASetOfCustomers*. Simultaneously, the export parameter `sales` is mapped to import parameter of the connector function `Add` (not shown in this model).

- Buisness object `Turnover Webshop 2` (CIM) is mapped to export parameter `Webshop2.getSales.sales` (PIM) because *TurnoverOfLastDay* is an upper concept of *TurnoverOfLastDayForASetOfCustomers*. Simultaneously, the export parameter `sales` is mapped to import parameter of the connector function `Add` (not shown in this model).

- Buisness object `Total Turnover` (CIM) is mapped to import paramter `ERP.setSales.sales` because *TurnoverOfLastDay* is an upper concept of *TurnoverOfLastDayForASetOfCustomers*. Simultaneously, export parameter of the connector function `Add` is mapped to the import parameter `sales`.

3.10 Scenario 5: Multiple Annotations

This scenario shall demonstrate the analysis of multiple annotations. These are used if a single annotation is not sufficient to convey the overall and complete semantic meaning of the annotated parameter or function. The scenario described in the following focuses on multiple representation annotation with the *AND*-operator (see section 5.4.2 for further description of the annotation type).

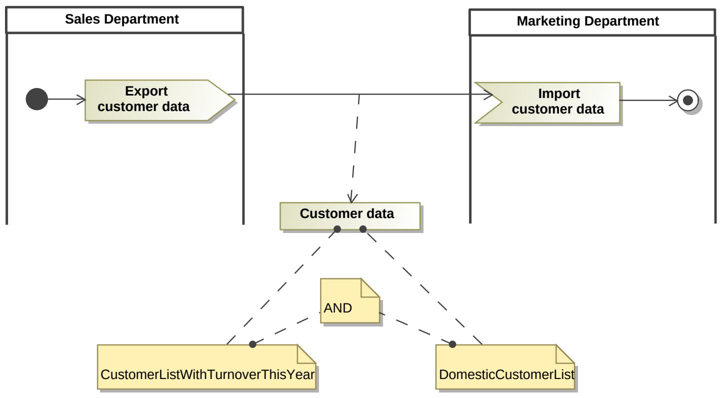


Figure 30: Scenario 5 - CIM level

At the CIM level (Figure 30) a very abstract business process is described. It defines the transfer of `Customer data` from the `Sales Department` to the `Marketing Department`, modeled as business components. The structure of the business object is not described any further, but the semantic annotation of it references the concepts *CustomerListWithTurnoverThisYear* and *DomesticCustomerList*, combined by the logical operator AND. The data transfered to the marketing department is an accumulated customer list, that includes customers that have purchased something in the past year, *and* have a domestic address.

The system's interfaces at the PIM level are shown in Figure 31. The technical realization of the Sales Department is represented by an ERP system, that offers two function interfaces `getDomesticCustomers` and `getCustomersWithTurnoverThisYear`. The export parameters of the interfaces deliver the domestic customer list and the list of customers, that made turnover the last year, respectively. Both parameters expose the same type structure, which is a set of customers composed by `id`, `firstName`, `lastName` and `emailAddress` fields. The Marketing System on the right side expects the same parameter type structure, but the import parameter `customers` is annotated with both concepts from the ontology, combined by logical AND.

The expected results of the semantic conflict analysis are:

- Interface `getDomesticCustomers` must be found for export (ERP, PIM)
- Interface `getCustomersWithTurnoverThisYear` must be found for export (ERP, PIM)

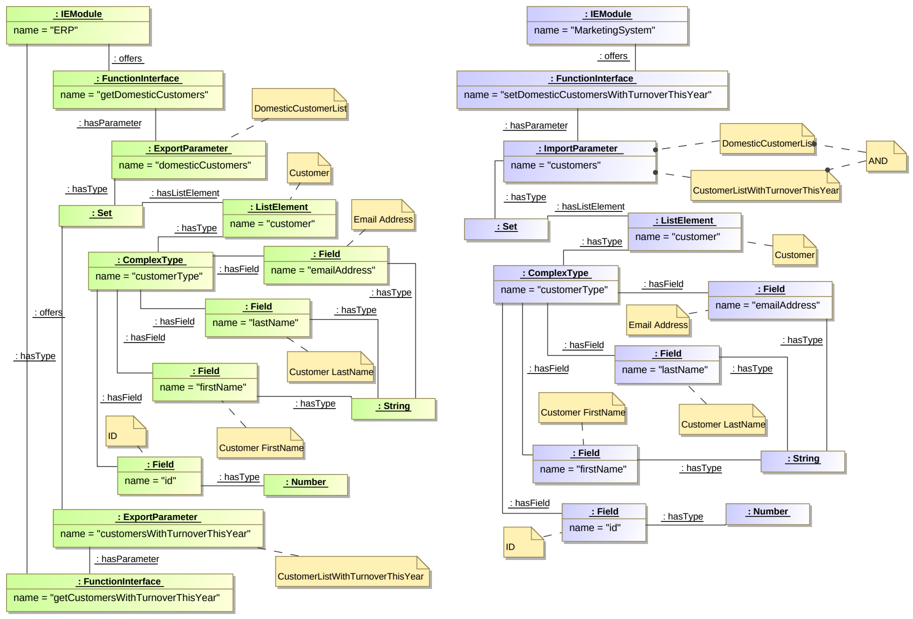


Figure 31: Scenario 5 - PIM level

- Interface `setDomesticCustomersWithTurnoverThisYear` must be found for import (MarketingSystem, PIM)
- Business object `Customer data` (CIM) is mapped to the import parameter `MarketingSystem.setDomesticCustomersWithTurnoverThisYear.customers` (PIM)
- SeCA will recognize the need for a connector function delivering a set of customers as export parameter, annotated with `DomesticCustomerList AND CustomerListWithTurnoverThisYear`. Since no implementation of such a function as available, the algorithm will suggest to implement a new one. The new connector function expects two import parameters annotated with `DomesticCustomerList` and `CustomerListWithTurnoverThisYear`, respectively.
- SeCA will map the import parameters of the new connector function to the export parameters `ERP.getDomesticCustomers.domesticCustomers` (PIM) and `ERP.getCustomersWithTurnoverThisYear.customersWithTurnoverThisYear` (PIM), respectively.
- Business object `Customer data` (CIM) is mapped to the export parameter of the new connector function (PIM).

3.11 Scenario 6: Integrating a Business Function

The following scenario describes the annotation and integration of a business function at the CIM level, and repeats the integration of a domain (helper) function at the PIM level. The CIM model (Figure 32) describes two business components, a Webshop and an ERP system. The Webshop exports a `Customer` object, which is annotated with *NetTurnover* and *InvoiceCountry*. The first concept denotes that the `Customer` object accumulates sum of all ordered items for a customer, and the second that the business object is uniquely identified with the country for which taxes subsequently have to be calculated. The ERP system performs exactly this: based on the net turnover and the tax rate for the invoice country, it calculates the respective taxes and generates the `Gross turnover` business object annotated with the concept *GrossTurnover*. Moreover, the business function at the ERP side, `Calculate gross turnover` is annotated with the concept *GrossTurnoverCalculation*. This function accepts a parameter annotated with *ValueAddedTaxRate* and *NetTurnover* and computes the object annotated with *GrossTurnover*.

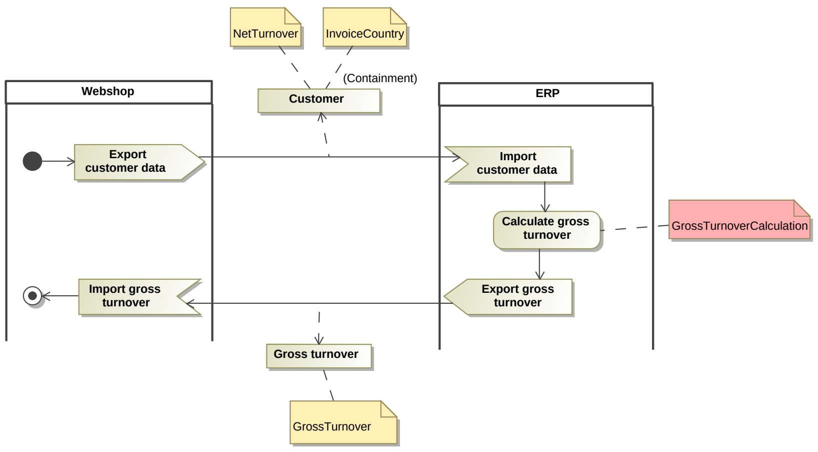


Figure 32: Scenario 6 - CIM level

At the PIM level (Figure 33) there are three `IntegratableElements` (Webshop, ERP and WebService) each with one interface. The Webshop offers interface `getCustomer` which exports `customer` object, with fields `country` and `netTurnover`, annotated with *InvoiceCountry* and *NetTurnover* respectively. Furthermore, the Webshop has the interface `setGrossTurnover` which expects a parameter `grossTurnover` annotated with *GrossTurnover*. The interface of the ERP system is, however, not able to accept the `customer` object generated by the Webshop directly, as its interface `calculateGrossTurnover` expects the `VATRate` parameter, which is annotated with *ValueAddedTaxRate*. In other words, the ERP system is not able to convert the invoice country to the corresponding tax rate for that country directly. Therefore, another interface is

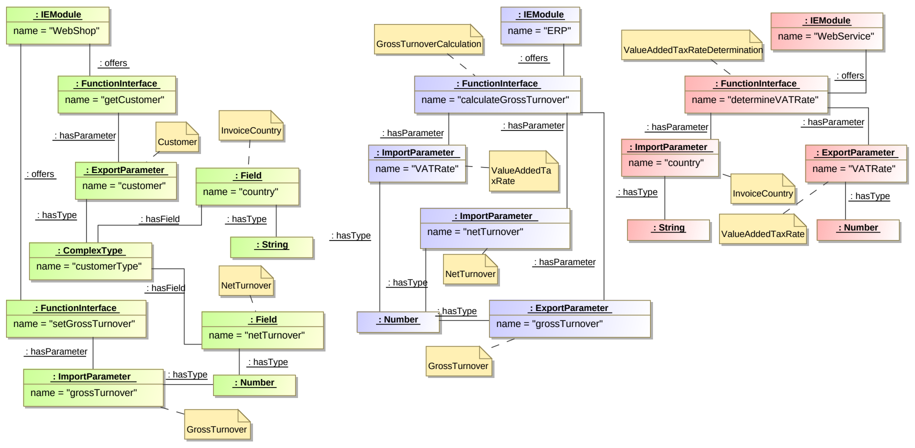


Figure 33: Scenario 6 - PIM level

modeled, a Web service which performs this conversion using the `determinVATRate` interface.

The expected results of the semantic conflict analysis are:

- Interface `getCustomer` must be found for export (Webshop, PIM)
- Business object `Customer` (CIM) is mapped to export parameter `Webshop.getCustomer.customer` (PIM), or more precisely to its fields `customer.country` and `customer.netTurnover` based on the containment annotation
- Interface `determineVATRate` must be found for export (ERP, PIM)
- Business object `Gross turnover` (CIM) is mapped to export parameter `ERP.calculateGrossTurnover.grossTurnover` (PIM)
- Business object `Gross turnover` (CIM) is mapped to import parameter `Webshop.setGrossTurnover.grossTurnover` (PIM)
- Interface `ERP.calculateGrossTurnover` must be found as a implementation for the business function `Calculate gross turnover`
- Import mapping for the business object `Customer` cannot be found. Therefore, additional interfaces are searched and `WebService.determineVATRate` is found for import.
- The export parameter `Webshop.getCustomer.customer.country` (PIM) is mapped to import parameter `WebService.determineVATRate.country` (PIM)
- The export parameter `WebService.determineVATRate.VATRate` is mapped to import parameter `ERP.calculateGrossTurnover.VATRate`

- The export parameter `Webshop.getCustomer.customer.netTurnover` (PIM) is mapped to import parameter `ERP.calcualateGrossTurnover.netTurnover`

Effectively, the scenario demonstrates how to annotate a business function at the CIM level, and to use a helper function to satisfy prerequisites for execution of the PIM interface corresponding to the CIM business function.

3.12 Scenario 7: Simultaneous Annotation of Business Objects and Business Functions

The following scenario shall demonstrate how business object requirements and business function requirements can be generated together and used for semantic reasoning. Due to the specific scenario requirements, we introduce a new business case (the CIM is given in Figure 34) as well as new domain ontology (Figure 35).

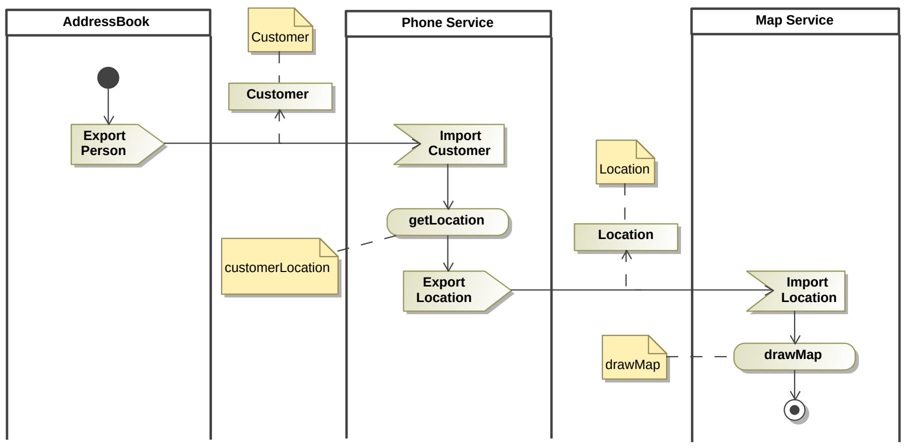


Figure 34: Scenario 7 - CIM level

The scenario describes integration of three systems: addressbook, phone provider and map service. The goal is to transfer customer information (business object `Customer`) from the addressbook to the phone provider ERP system, determine the phone (and customer's) location, and then draw a map with the current phone/user location using a map service. The PIM interfaces realizing this scenario are given in Figure 36.

The `Addressbook` module is a database system which exports the `Customer` object. The ERP system `phoneService` of the mobile phone provider offers two functions with the same signature and semantic parameter annotations, `getPhoneLocation` and `getSupprotLocation`. This example shows that data annotations are in some cases not expressive enough, as these two functions return an object which has the *Location* semantic, but one location refers to the location of the phone itself, and the other to the location of the nearest

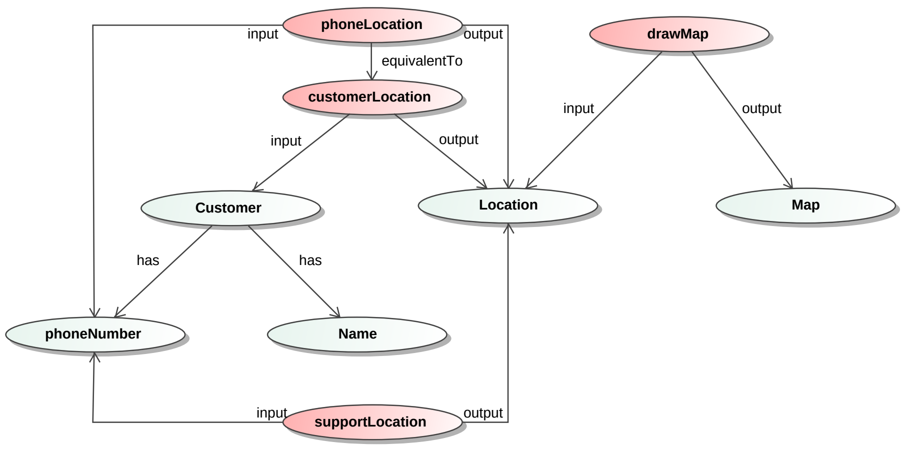


Figure 35: Scenario 7 - Ontology

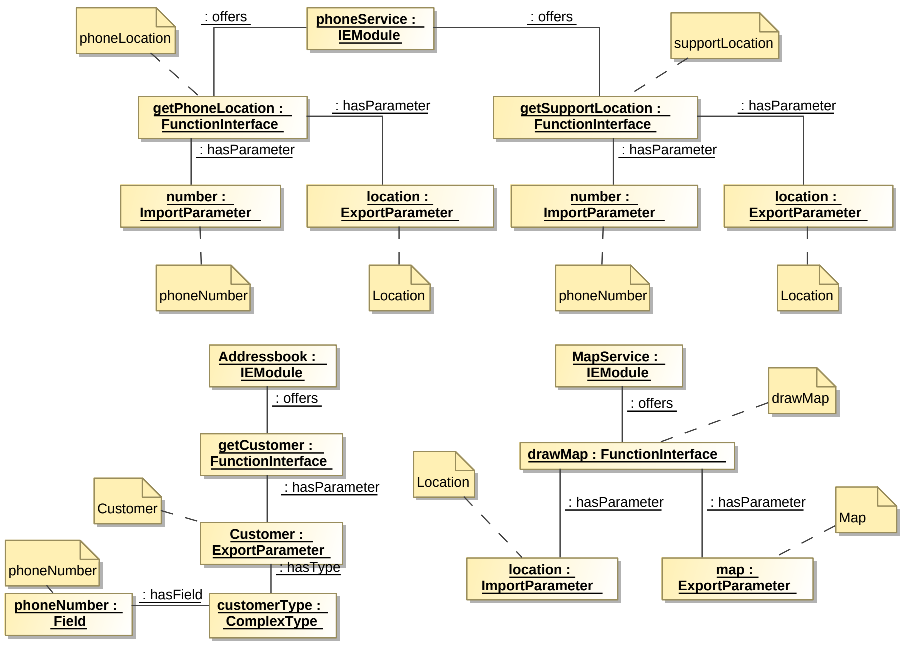


Figure 36: Scenario 7 - PIM level

support office. This distinction is expressed using two additional functional annotations, *phoneLocation* and *supportLocation*. Finally, the Web Service `Map Service` is used to draw a map with the current customer location. The main problem in this scenario is how to deduce that the location of a customer may be determined via location of his/hers mobile phone (business process at the CIM level requires customer location, but there is no technical interface realizing this functionality directly) and how to use functional annotations to select between functions which offer the same structural signature (computing the phone and support office locations) but have different meaning. To that end, the algorithm shall perform the following:

Variant A(annotations as given in Figure 36)

- The interface `getCustomer` (PIM) must be found for export.
- Business object `Customer` (CIM) is mapped to export parameter `Address-book.getCustomer.Customer` (PIM).
- The import mapping for the business object `Customer` cannot be found. Interfaces `phoneService.getPhoneLocation` and `phoneService.getSu-pportLocation` must be found for import. The field `Customer.phoneNumber` (PIM) is then mapped to `phoneService.getPhoneLocation.number` and to `phoneService.getSupportLocation.number` (PIM).
- Business object `Location` (CIM) is mapped to export parameters `phone-Service.getPhoneLocation.location` (PIM) and `phoneService.getSu-pportLocation.location` (PIM).
- Business object `Location` (CIM) is mapped to import parameter `MapSer-vice.drawMap.location`.
- Export parameters `phoneService.getPhoneLocation.location` (PIM) and `phoneService.getSupportLocation.location` (PIM) are mapped to import parameter `MapService.drawMap.location` (PIM).
- The algorithm has found two solutions for calculating location. Using business function requirements this choice will be reduced to a single function. Business function `customerLocation` cannot be directly mapped. Using reasoning and the domain ontology, the function `getPhoneLoca-tion` is found which is annotated with *phoneLocation* – the location of a person can be determined by locating his/hers mobile phone (see the F-logic example below which demonstrates how this equivalence is proven, the assumption is that a person always carries a mobile phone). Function `getSupportLocation` annotated with *supportLocation* is not semantically compatible with the *customerLocation* concept, therefore all business object mappings already generated for this function are now removed (`Customer.phoneNumber` to `getSupportLocation.number` and `Location` to `getSupportLocation.location`). Thus a single solution is obtained.
- Business function `drawMap` (CIM) is mapped to `MapService.drawMap` (PIM).

To illustrate the reasoning process, we also give excerpt from the formalized ontology representation using F-logic notation:

```
parameter[].
Location:parameter.
phoneNumber:parameter.
Customer:parameter[Name *=> string, phone *=> phoneNumber].
phoneLocation(phoneNumber,Location).
customerLocation(Customer,Location).
supportLocation(phoneNumber,Location).
customerLocation(C,L) :- X[phoneNumber -> P], P:phoneNumber,
C:Customer, phoneLocation(P,L).
```

The last rule says that the location L of a customer C is equivalent to determining the phone number P of the same customer, and then determining the location L of that phone number.

Variant B: Parameters of the interface `phoneService.getPhoneLocation` have no semantic annotations. The algorithm still offers the function `getPhoneLocation` as the potential correct solution, requires however manual confirmation through parameter annotation before proceeding further. Function `phoneService.getSupportLocation` is eliminated as in the previous variant.

Variant C: Interface `phoneService.getPhoneLocation` is split into two interfaces, `setPhone(phoneNumer)` and `location calculateLocation()`. Both are compositely annotated with functional annotation *phoneLocation*. The algorithm shall discover that both functions have to be invoked and furthermore it will discover the call order using the input and output parameter annotations.

Variant D: Interface `phoneService.getPhoneLocation` is split into three interfaces, `setPhone(phoneNumber)`, `calculateLocation()` and `location getLocation()`, where only `calculateLocation` is annotated with *phoneLocation*. The algorith shall disover that all three functions have to be invoked, but it will fail to determine the correct call order, if parameters are not additionally annotated.

Variant E: If no functional annotations are present, the algorithm shall present the choice between two compatible functions `getSupportLocation` and `getCustomerLocation`.

Variant F: If the ontology is modified in such a way that it is not possible to prove the equivalence of *phoneLocation* and *customerLocation* concepts, the algorithm shall yield the unresolvable semantic conflict.

3.13 Summary of the Semantic Conflict Analysis Requirements

Based on the presented scenarios, the following requirements for the semantic conflict analysis algorithm can be summarized:

- The algorithm shall enable mapping between CIM and PIM levels, effectively checking if business process requirements are semantically matched at the PSM/PIM level.
- The algorithm shall be able to check semantic matching between simple (unstructured) business objects and parameters.
- The algorithm shall be able to check semantic matching between complex (structured) business objects and parameters using decomposition.
- The algorithm shall be able to filter out irrelevant as well as optional data.
- The algorithm shall be able to discover semantic dependencies between multiple interfaces and calculate correct invocation (call) order.
- The algorithm shall be able to detect semantically equivalent possibilities to satisfy CIM to PIM as well as PIM to PIM requirements. It shall furthermore offer backtracking to all semantically correct but discarded parameter and interface combinations if subsequent conflict analysis procedures invalidate the scenario.
- The algorithm shall be able to recognize a deadlock in interface call order based on semantic annotations.
- The algorithm shall be able to recognize collections of objects or parameters and perform semantic matching between collection element and single objects. For that purpose the algorithm shall generate collection iterators.
- The algorithm shall support integration of domain functions, if such are included in the ontology.
- The algorithm shall support integration of connector function.
- The algorithm shall be able to import and read ontologies developed under the ontology metamodel (see Section 1.1.5).
- The algorithm shall be able to use reasoning on the existing ontology to discover relationships between concepts. Reasoning on multiple annotations connected using logical operators shall be supported.
- The algorithm shall support integration of the business functions.

Based on these requirements, the semantic conflict analysis algorithm will be formally defined in Section 6.1.

4 Conflict Analysis Algorithm

The BIZYCLE integration platform supports software and data integration process through conflict analysis and resolution. It recognizes different types of integration conflicts and suggests solutions. It is aimed to automate the process, however there will be the cases where automatic detection and/or resolution is not possible. In such cases, notification to the integration specialists is performed.

As information related to system integration is abstracted at the platform independent level (PIM), it is possible to analyze different types of integration conflicts regardless of the underlying technical platforms.

4.1 Conflict Types

The conflict analysis process described in this report is primarily based on platform independent component descriptions, i. e. on the instances of the platform independent metamodel (PIMM) introduced in Section 1.1.4. The following integration conflict types can be distinguished:

- *Semantic conflicts* address the semantical aspects of modeled integratable elements. Semantic conflicts occur when compared entities that should be integrated (interfaces, data or business objects, parameters etc.) differ in the meaning they convey, that is, when semantically incompatible entities are treated as equal, based on other properties such as their structure.

- *Behavior conflicts* concern the dynamic aspects of integratable elements and are primarily caused by function constraints. For instance, a violated pre-condition can cause a behavior conflict which would point at an invalid function call order or incorrect functional capability.

- *Property conflicts* pertain to the characteristics of the integratable elements. Most notably, these are Quality of Service (QoS) properties (performance, reliability, security etc.) in conjunction with several metrics (duration, periodicity, units etc.), which describe requirements specified by components to be integrated as well as requirements these components have to fulfill.

- *Structure conflicts* concern the static aspects of the component description and are caused by differences in data structures, i. e. data types and runtime values. It is important to note that both the analysis and resolution of structural conflicts are based on the design time description of values, not on the actual values at the runtime.

- *Communication conflicts* occur when the integratable elements differ in communication aspects, e.g., use conflicting communication protocols or incompatible kinds of messaging. The communication properties of a system depend on a platform the system is running at. Communication conflicts are resolved by generation of appropriate application endpoints.

With respect to the solution possibilities and its degree of automation, we distinguish the following types:

- *Automatically resolvable conflicts* can be resolved by a conflict analysis tool without user interaction. In this case, the conflict resolution process provides a deterministic solution plan to eliminate the conflict.

- *Non-automatically resolvable conflict* require at least one user interaction to be resolved. There is at least one possible solution, but the conflict resolution process contains a non-deterministic plan to eliminate this conflict. We also can say, the conflict is *semi-automatically resolvable.*

- *Non-resolvable conflicts* can be eliminated neither automatically nor semi-automatically. In this case, the conflict analysis process should be aborted. Non-resolvable conflicts can be caused not only by the incompatibility of two integratable element but also by their incomplete description.

4.2 Overview of the Conflict Analysis Algorithm

In this section we give an overview of the conflict analysis and resolution algorithm. We introduce in short each of the process phases and specify the artifacts used and generated by the conflict analysis tool when executing these phases. The conflict analysis process consists of five consecutive analysis phases which handle conflicts of the corresponding type. The transition to the next phase is possible only if in the current phase no non-resolvable conflicts could be detected. The complete conflict analysis process is shown In Figure 37.

Performing the semantic conflict analysis in the first phase is plausible because the results of following analysis phases are meaningful only if the components to be integrated are semantically conflict-free. The semantical analysis phase uses following artifacts as inputs:

- Platform independent models (PIM) of integratable elements

- CIM model describing the integration scenario and the list of defined requirements generated on its basis

- Domain ontology

The semantic analysis phase generates a list of requirement mappings as output. Requirement mappings assign business requirements described in CIM to the technical interfaces of the integratable element modeled in PIM in terms of semantical aspects. However, the semantic conflict analysis is optional, e. g., in case when appropriate ontology is not available. The integration specialist may skip this phase and specify the appropriate requirement mappings manually.

The next two phases, behavior and property conflict analysis, are performed in parallel. The reason is that descriptions of the functional constraints (behavior) refer to the QoS properties and metrics defined in the Property package of the PIMM. The inputs of the behavior and property conflict analysis phase are:

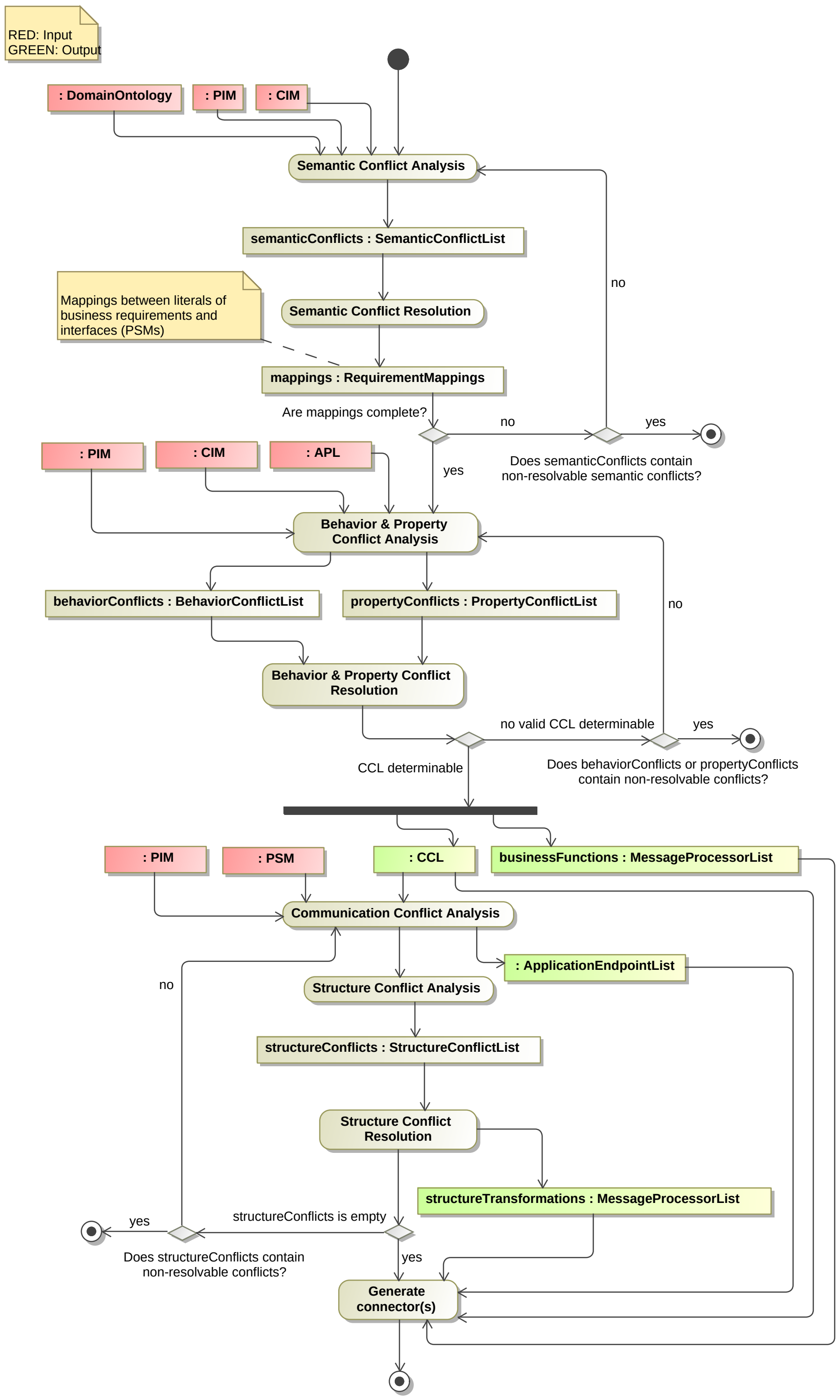

RED: Input
GREEN: Output
: DomainOntology
: PIM
: CIM
Semantic Conflict Analysis
semanticConflicts : SemanticConflictList
Mappings between literals of business requirements and interfaces (PSMs)
Semantic Conflict Resolution
mappings : RequirementMappings
Are mappings complete?
no
yes
no
Does semanticConflicts contain non-resolvable semantic conflicts?
yes
: PIM
: CIM
: APL
Behavior & Property Conflict Analysis
behaviorConflicts : BehaviorConflictList
propertyConflicts : PropertyConflictList
no
Behavior & Property Conflict Resolution
no valid CCL determinable
yes
Does behaviorConflicts or propertyConflicts contain non-resolvable conflicts?
CCL determinable
: PIM
: PSM
: CCL
businessFunctions : MessageProcessorList
Communication Conflict Analysis
: ApplicationEndpointList
no
Structure Conflict Analysis
structureConflicts : StructureConflictList
Structure Conflict Resolution
structureTransformations : MessageProcessorList
yes
structureConflicts is empty
Does structureConflicts contain non-resolvable conflicts?
yes
Generate connector(s)

Figure 37: Conflict Analysis Process

- Platform independent models of integratable elements
- Computational independent model
- Application Protocol Logic (APL) containing a set of execution paths for interfaces involved in the integration scenario
- Requirement mappings from the previous analysis phase

The behavior analysis checks the control and data flows specified in CIM as well as the results of the semantic conflict analysis (requirement mappings) against the behavior constraints described by APL which was previously created by the application specialist. The goal is to determine a conflict-free call order within the connector logic, which would fulfill the business requirements defined in CIM. As a result the behavior and property conflict analysis phases deliver following artifacts:

- Connector Call Logic (CCL) containing a conflict-free interface (and message processor) call order the connector should implement
- List of message processors which provide support for business solutions

The next phase, communication conflict analysis, deals with communication properties of integratable elements. It requires platform-specific data delivered by PSM. The results of this phase provide a basis for generation of application endpoints.

The final phase of conflict analysis process is the structural conflict analysis, addressing data type structure and the runtime value format to overcome structural heterogeneity. If the conflict resolution process succeeds (that means each detected conflict were resolved), CCL, message processor list and application endpoint list are used for connector generation.

5 Semantic Annotation

Within the BIZYCLE integration process, models at different levels of abstraction are created to enable the (semi-)automatic connector generation. The integration scenario with its abstract flow and data model is specified with the computation independent model (CIM). Technical interfaces are modeled in platform specific models (PSMs) that are transformed to platform independent models (PIMs) for comparison and integration conflict analysis.

To further support an automated detection of interface conflicts we propose semantic description of the integration scenario models at all levels of abstraction. In section 1.1.5 the ontology metamodel was introduced. The ontology contains knowledge of the integration domain that is shared among different integration models or integration projects. It can be refined and extended during the projects. The association of heterogeneous artifacts such as documents, service interfaces, business processes, web resources and models with semantic concepts is called semantic annotation in general [34][20]. In the context of the BIZYCLE integration, semantic annotation is a process of creating relationships between the integration models at CIM, PSM, PIM level and the ontology concepts. At meta-level an annotation metamodel (AMM) was developed, linking all other metamodels within the BIZYCLE project to the semantic metamodel and also supporting the integration of existing external platform specific metamodels by adding associations between appropriate metaclasses. The advantage of the intermediate metamodel is that all instantiated models are independent of the semantic metamodel. Changes to models or even metamodels are managed at a central point. This is approach similar to the model weaving [23].

Figure 38 gives a short overview of the relations between different metamodels that are involved in an integration project. In the following sections it is described which kind of semantic annotations are supported, which model elements at all abstraction levels can be annotated with ontology concepts and how the semantic annotations can be used and combined.

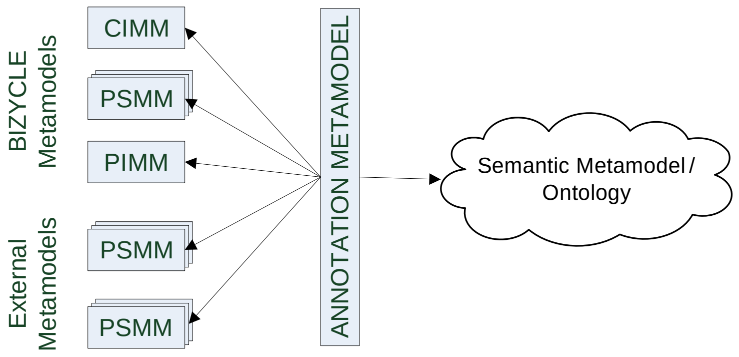


Figure 38: Semantic annotation metamodel relations

5.1 Data-oriented Annotations

The annotation metamodel supports two kinds of annotations: data-oriented and function-oriented (see Section 5.2). The former is used to describe all model elements that represent data at a certain level of abstraction. A knowledge concerning data objects is expressed by creating instances of the `DomainObject` (Section 1.1.5 or 3). A data-oriented annotation is defined as `DomainObjectAnnotation` metaclass. On one side, all data-related metaclasses at the CIM, PSM and PIM level are referenced (annotation source). On the other side it refers to the `DomainObject` metaclass of the semantic metamodel (annotation target). The relationships at different abstraction levels are described in the following sections.

5.1.1 CIM Level Annotations

At the most abstract modeling level (CIM), data objects and their structure are defined using the metaclass `BusinessObject` (CIMM). The annotation metaclass of the AMM links it to the `DomainObject` of the SMM (Figure 39).

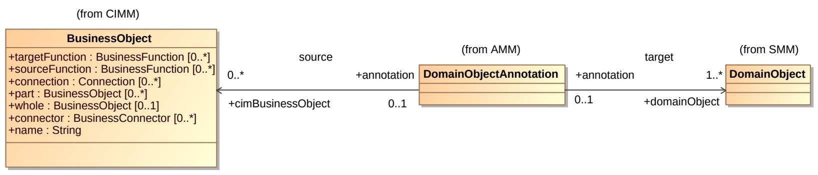

(from CIMM)
BusinessObject
+targetFunction : BusinessFunction [0..*]
+sourceFunction : BusinessFunction [0..*]
+connection : Connection [0..*]
+part : BusinessObject [0..*]
+whole : BusinessObject [0..1]
+connector : BusinessConnector [0..*]
+name : String
source
0..*
+annotation
+cimBusinessObject
0..1
(from AMM)
DomainObjectAnnotation
target
+annotation
0..1
1..*
+domainObject
(from SMM)
DomainObject

Figure 39: Semantic data annotation at the CIM level

5.1.2 PSM Level Annotations

At the PSM level, data annotations are provided for internal as well as external metamodels. In the following sections we will discuss how annotations are realized for exemplary platform specific metamodels.

PSM level annotations – SAP R/3
The platform specific metamodel for the SAP R/3 systems allows the modeling of several data-oriented aspects of the SAP R/3 interfaces. An application specialist can specify Business API functions (BAPI), that contain input and output `Parameter` elements which are typed according to their content (tables, structures and simple types such as strings or integers). Furthermore SAP R/3 systems offer document interfaces in terms of structured `IDOCType` with `DataRecord`, `Segment` and typed `Field` elements. All of these metaclasses are linked to the annotation metamodel to be able to declare the meaning of the interface structure at respective levels of refinement. Figure 40 shows the relations. For better readability all internal associations between SAP elements have been removed.

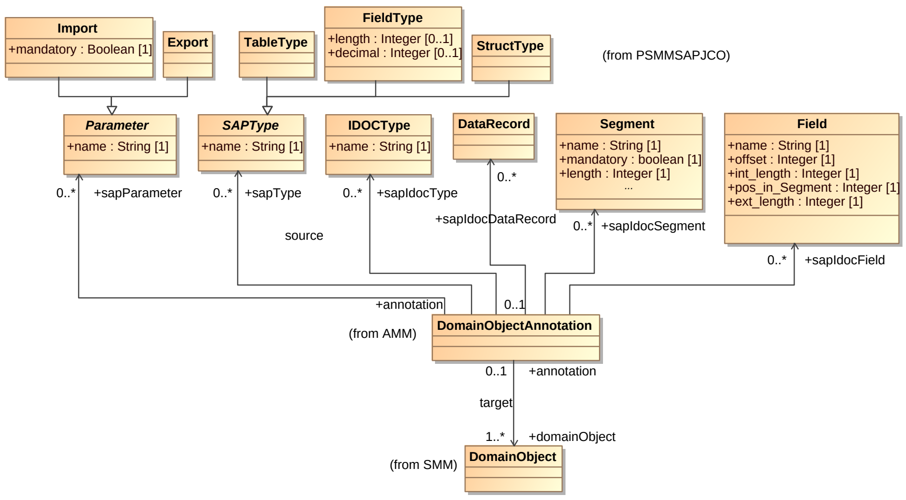


Figure 40: Semantic data annotation at the PSM level – SAP R/3 systems

PSM level annotations – J2EE

Another platform that is supported by BIZYCLE are EJB components running in J2EE containers. They offer Enterprise JavaBeans with methods, that have `Input` and `Output` parameters which are subject to semantic annotation. The parameters have simple types (such as floats and integers), complex types composed of `Field` elements as well as collections. In Figure 41 the annotation associations to the J2EE metamodel are shown.

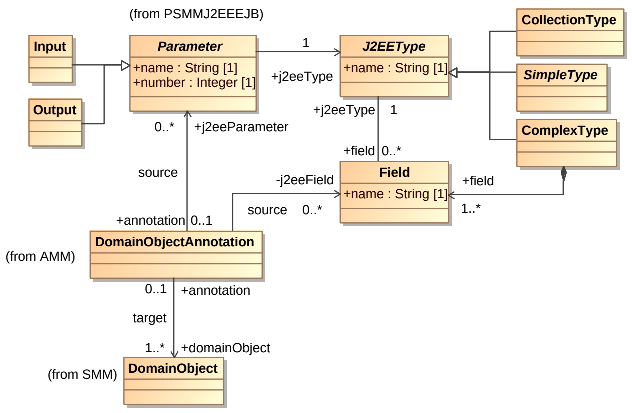


Figure 41: Semantic data annotation at the PSM level – J2EE systems

PSM level annotations – RDBMS

The metamodel for relational database management systems offers the `Query` metaclass to describe a set of `Input` or `Output Parameters` as well as their relationships. Parameters have a type, such as the JDBC interface types CHAR or DATE. Input and output parameter sets corresponding to the database attributes or aggregated and computed fields in record sets are semantically annotated as shown in Figure 42.

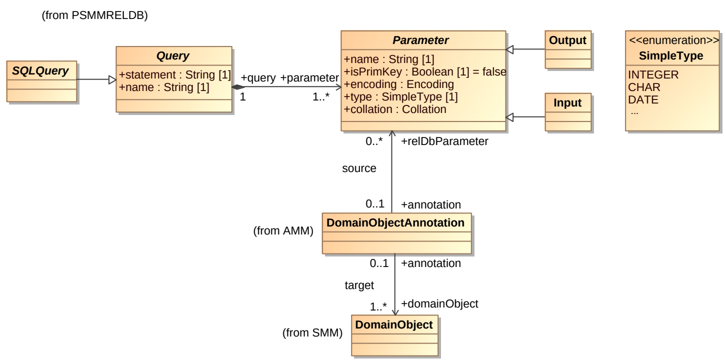


Figure 42: Semantic data annotation at the PSM level – RDBMS

PSM level annotations – external metamodels

In section 5 we have introduced the possibility of integrating external metamodels for semantic annotation. As an example we present the annotation of the XSD metamodel. With the XSD metamodel it is possible to define XML schemas that describe the structure of XML documents. Usage of the metamodel is necessary for data integration of XML files and for the description of web service interfaces (WSDL). In Figure 43 a small excerpt of the XSD metamodel is shown. We currently identified `DocumentRoot`, `Group` and `Element` as annotatable elements. Since the AMM and SMM is implemented as Ecore metamodel it is an easy task to model the relevant metaclasses of an external metamodel and replace the according references to an existing Ecore implementation.

5.1.3 PIM Level Annotations

The annotations illustrated in the previous sections enable semantic description of data oriented interface elements at the PSM level. To carry out the conflict analysis, a model-to-model transformation of the PSM models to PIM models is performed. Semantic descriptions given at the PSM level are thus transformed to the PIM level as well. The AMM offers the appropriate associations to the PIMM. The PIMM encapsulates all data elements that can be annotated with the `AnnotatableElement` metaclass. The specialized elements are `OOAtribute`, `OOParameter` and `TypedElement`, such as parameters and fields of complex types. With the reference from the AMM to the `AnnotatableElement` it is possible to describe the precise meaning of every interface and deliverable and/or consumable data on the abstracted PIM level.

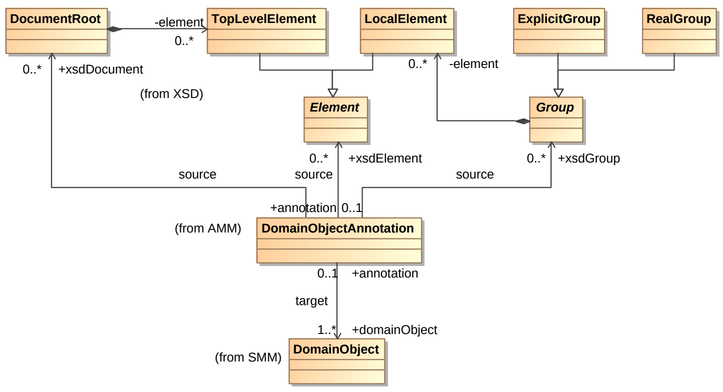


Figure 43: Semantic data annotation at the PSM level – External XSD metamodel

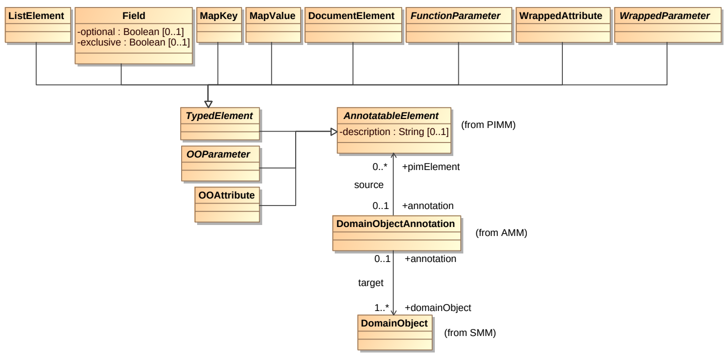


Figure 44: Semantic data annotation at the PIM level

5.2 Function-oriented Annotations

The AMM offers function-oriented annotations, too. These annotations are aimed at describing the meaning of the computational and processing behavior of the elements such as remote functions, database queries or Web service methods. The class `DomainFunction` from the semantic metamodel is used to assign this meaning to a model element. The AMM establishes a relationship between a functional element at the CIM, PSM or PIM level and the domain function from a given ontology using the `DomainFunctionAnnotation` element.

Figure 46 shows the elements from the CIM, PSM and PIM levels that may be annotated using functional semantics. At the CIM level, `BusinessFunction` and `ConnectorFunction` convey the processing semantics of the integratable systems and connectors respectively. Thus they are annotated with domain function annotations.

At the PIM level, all three supported interface types, `FunctionInterface`,

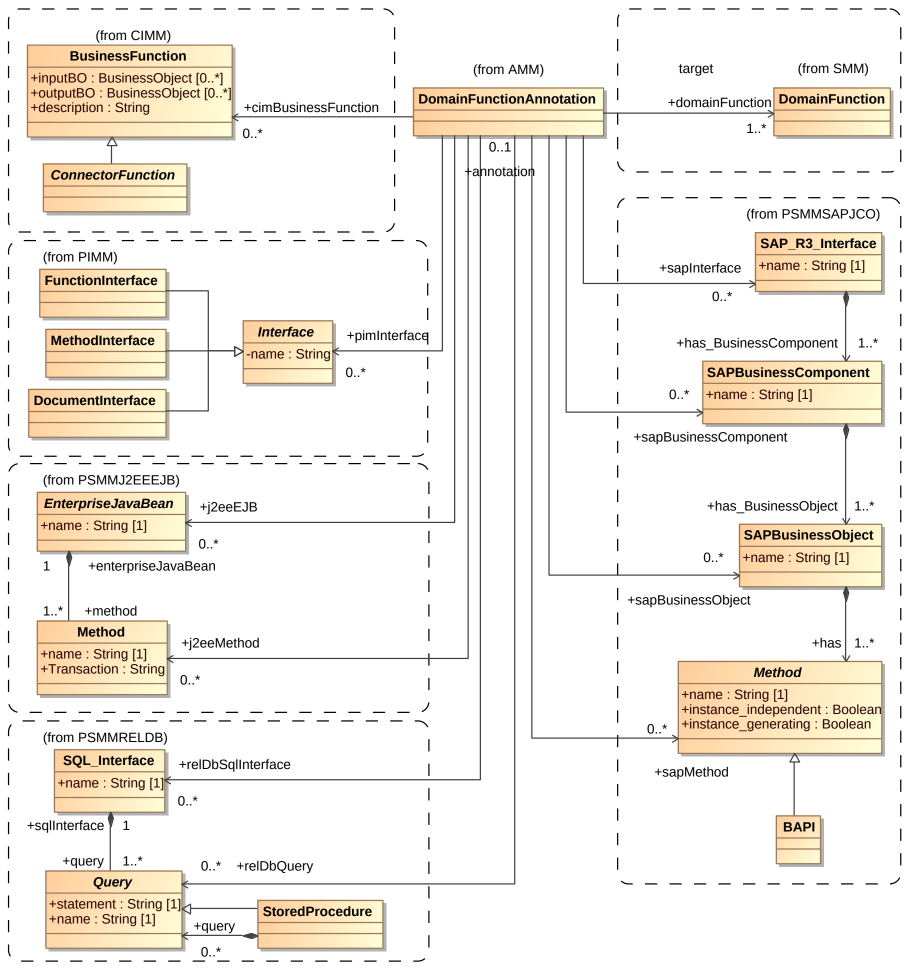


Figure 45: Functional semantic annotation

`MethodInterface` and `DocumentInterface` are annotatable via domain function annotations. At the PSM level, functionally-annotatable metaclasses differ, depending on the metamodel. In SAP R/3 systems, the following metaclasses have the processing semantics, and are thus functionally annotated: `SAP_R3_Interface`, `SAPBusinessComponent`, `SAPBusinessObject`, `Method` and `BAPI`. Note the varying degree of granularity of these metaclasses, allowing to specify functional semantics of more or less entire SAP interface on one side, as well as single remotely accessible BAPI function on the other. The J2EE metamodel exposes similar possibility: either the entire bean can be functionally annotated as the source of the application functionality, or the separate remote methods can be individually annotated. Finally, for the relational database management system, we treat queries as functionality carriers, and allow the functional annotation of SQL queries and stored procedures accordingly.

5.3 Logical Operators

Apart from the annotation metaclasses, the AMM also offers means to combine (group) annotations using logical operators. Depending on the refinement of the ontology and the specified integration scenario at the CIM and PSM/PIM level, business architects, application specialists and integration specialists must have the possibility to annotate a model element with more than one meaning or to annotate several model elements together with one concept from the ontology. The annotation metamodel offers three logical operators (AND, OR, XOR) as shown in Figure 46.

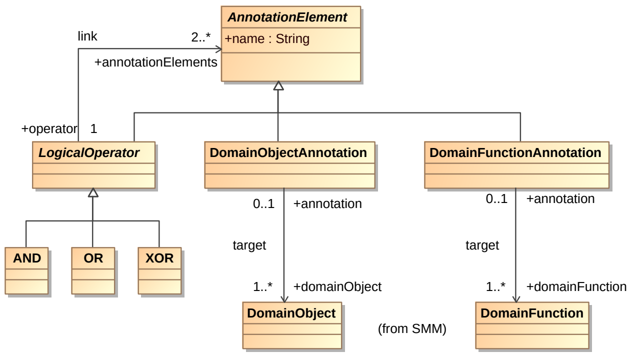


Figure 46: Annotation metamodel core and logical operators

The metaclass `LogicalOperator` links two or more `AnnotationElement`, that can either be annotations or other operators. Thus it possible to create nested composition of operators such as *(Identifier)* || *(First name* && *Last name)*. The association `link` is navigable from the operator to the annotation element only, to determine the nesting direction.

5.4 Applications of Semantic Annotation

The logical operators described above, the 1..* cardinality of a semantic annotation to the semantic metamodel elements and the references to all supported metamodels (CIMM, PSMM, PIMM), allow for various combinations of semantic descriptions with a large degree of expressiveness. It is possible to build containment, alternatives, composition and multiple representation expressions. Furthermore, traceability of information is supported by linking model elements of different abstractions levels. The linkage of different types of models must be used carefully, because an instantiation of the annotation metaclasses could lead to unreasonable combinations of model elements. The following source link combinations (see previous sections) are allowed:

- Multiple source references of a single annotation are limited to elements of the same model (either CIM, or PSM, or PIM), except for *PSM and PIM* links, if the PIM elements have been obtained from respective PSM elements by model-to-model transformation (see section 5.4.6).
- The combination of multiple annotations by logical operators is limited to elements of the same source model (either CIM, or PSM, or PIM).

In the following sections a short overview of the relevant annotation alternatives is described.

5.4.1 Single Representation Annotations

The simplest case of the semantic annotation is the 1:1 relationship of a model element with a concept of the ontology. Figure 47 illustrates a CIM business object annotation as an example. At the right side an excerpt of an ontology is shown, that defines the domain object *Address*, which consists of several sub-concepts (*Lastname*, *Street*, etc.). In the annotation model one domain object annotation is created, that links `Shipping address` at the CIM level to the *Address* concept from the ontology. To reduce the effort of annotation the link means that the shipping address at the CIM level is an address and implies all of the sub-concepts that are associated with *has*-predicates in the ontology.[3] If the annotation should be more precise (e.g., individually selecting some of the sub-concepts), either containment annotation or business object refinement is necessary.

The single representation annotation is applicable for data-oriented and function-oriented annotations of the CIM and PSM/PIM models.

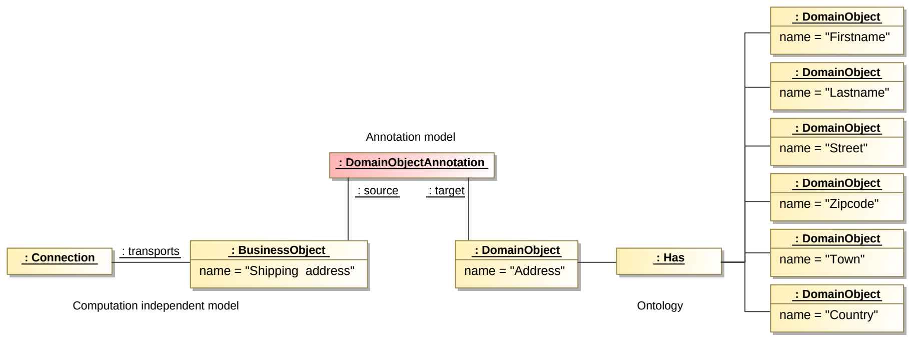


Figure 47: Annotation with a single domain object representation

5.4.2 Multiple Representation Annotation

Multiple representations occur when a certain model element has more than one meaning at the same time (logical conjunction). Figure 48 shows an example

[3]The implementation of the annotation component should leave that at the user's choice.

of the interface annotation at the PIM level. The function interface exposes an import parameter `shopCust`, that serves as input to retrieve a customer object from the Webshop system. The Webshop system expects a parameter that represents both *CusomerName* and *CustomerID*, which means that it has a semantic of a name, but at the same time of a primary key (identifier). An example may be a VARCHAR field from a database that is defined as primary key. In the example two annotations are created to link the parameter to the appropriate domain objects of the ontology. After that, both annotations are linked with the AND operator. That results in the expression *CustomerID && CustomerName*. Functional annotations can also be combined in such a way that a single interface represents different functionality depending on its parameters.

The multiple representation annotation is applicable for data-oriented annotation of CIM (limited) and PSM/PIM models.

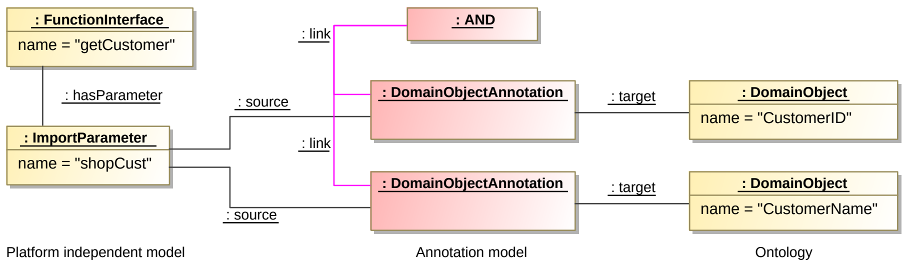


Figure 48: Multiple representation annotation

5.4.3 Containment Annotation

The containment annotation is used when a certain model element has a rough level of detail or does not have a representation in the ontology, but it can be composed of several concepts. It can also be used to select a set of sub-concepts if the single representation annotation (see section 5.4.1) is not adequate. Figure 49 shows an example of the containment annotation. It is realized by creating a single domain object annotation with multiple references to the domain objects. The expression means that the `Shipping address` defined in the CIM level is composed of *Firstname, Lastname, Street, Zipcode, Town* concepts.

As an alternative the content description at the CIM level can be achieved by explicitly creating structured business objects and using the single representation annotation. At the PSM/PIM level such a refinement is not possible because of the immutable interface structure of the underlying systems, so the containment annotation can be used when needed.

An implementation of the containment annotation must assure that the multiple references of the domain object annotation do not link concepts of different hierarchies, e.g., a containment annotation with *Address* and *Firstname* at the

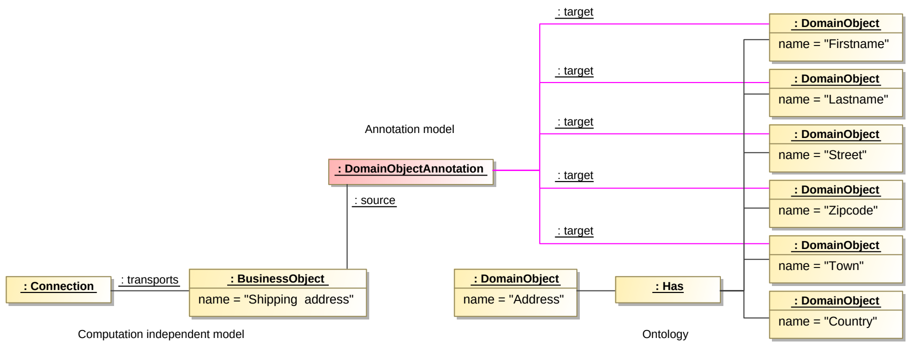


Figure 49: Containment annotation

same time is not allowed.

The containment annotation is applicable for data-oriented and function-oriented annotation of CIM and PSM/PIM models.

5.4.4 Alternative Annotation

Alternative meanings (logical disjunction) of model elements is mainly observed in interface descriptions that have a fixed output data structure, but the data exchanged through the interface varies depending on input parameters. Secondly, it can also signify mixed content, that does not have a single knowledge representation. Both cases are annotated using multiple annotations linked with the *OR*- or *XOR*-operator. In Figure 50 an example is illustrated in which the export parameter `customers` depends on the input parameter `flag`. To semantically express that either *ActiveCustomers* or *InactiveCustomers* are returned, the XOR-operator is used. If the OR-operator is used, the semantic descriptions means that both active and inactive customers may be returned.

The alternative annotation is applicable for data-oriented and function-oriented annotation of CIM and PSM/PIM models.

5.4.5 Compositional Annotation

The compositional annotation can be used if a model and its elements have a high level of detail and a direct match to ontology elements is not possible. In that case several model elements are considered together as a single knowledge representation. A compositional annotation is created with a single annotation instance, which references multiple source model elements. An example of such annotation is depicted in Figure 51. It shows a part of a J2EE system with an Enterprise JavaBean offering three methods. Together they realize the summation of a price list, but the interface structure requires separate consecutive calls.

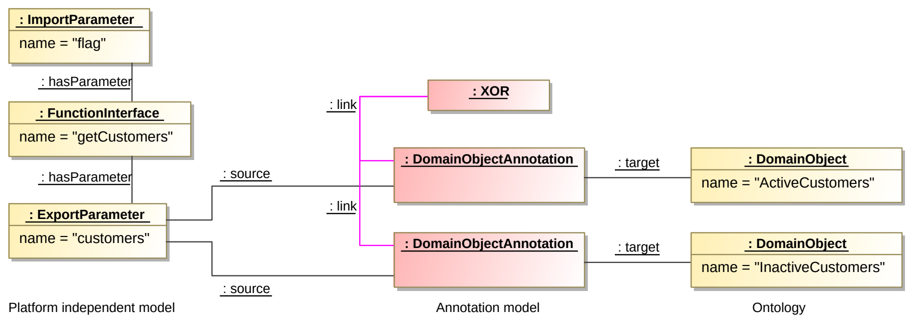


Figure 50: Alternatives annotation

The compositional annotation is applicable for data-oriented and function-oriented annotation PSM/PIM models.

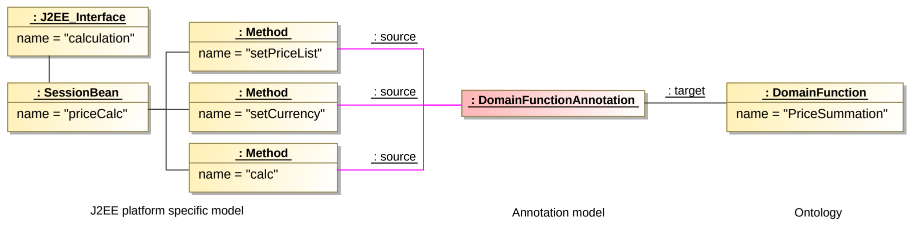


Figure 51: Compositional annotation

5.4.6 Multilevel Annotation

In the previous sections we have described the possibilities of semantic annotations that are mainly used to enable the semantic conflict analysis. The usage of annotations proposed so far is limited to single abstraction levels. Additionally the annotation metamodel provide means for the traceability of information. In the following an example is described that makes use of linking model elements on different levels of abstraction for the purpose of establishing transformation traces.

During the BIZYCLE integration process platform specific models are transformed to platform independent level to perform the conflict analysis. The semantic descriptions of the interfaces are created at PSM level, therefore they have to be transfered to PIM level as well. During the model-to-model transformation the annotation model is used as second input. The transformation rules create all PIM model elements according to the PSM. For each annotated PSM element, a reference to the corresponding PIM element is added to the annotation element. The transformation not only creates a platform independent model, but enriches (extends) the annotation model. Figure 52 illustrates

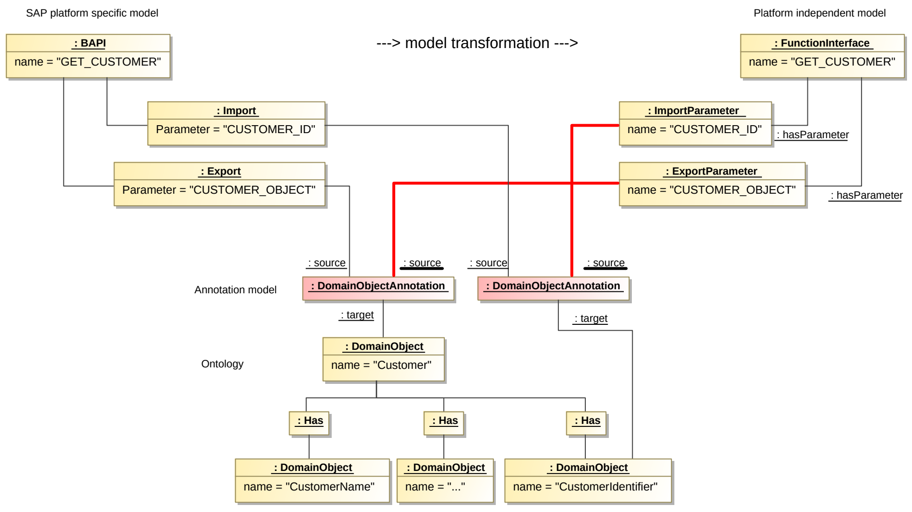


Figure 52: Semantic annotation for model transformation traces

an example of the SAP model transformation. Before the transformation, the SAP import and export parameters on the left side are annotated with the corresponding domain objects. The references to newly added PIM elements (`ImportParameter` and `ExportParameter`) are created during the transformation. Later, the PIM import parameter can be identified as being generated from the SAP BAPI import parameter through the domain object annotation. An identification based on names is not distinct. Furthermore, the transformation traces help application specialists to recognize the system's models during decisions in the conflict analysis.

6 Semantic Conflict Analysis

In this chapter, the semantic conflict analysis algorithm will be formally described. After that a metamodel is introduced, that is used to represent the results of the analysis. Furthermore, analysis examples will be given. Finally, as the algorithm is described in the language-independent manner using the UML activity diagrams, the chapter concludes with the overview of the current Prolog implementation.

6.1 Algorithm Description

In the previous chapters the premises for the semantic conflict analysis have been described. To sum up, the following artifacts (models and relations) have to be available for the semantic conflict analysis:

- Ontology, containing semantic definitions from a domain that is relevant for the integration scenario
- Computation independent model, describing the scenario and its requirements
- Platform independent models of the systems to be integrated, obtained by PSM transformations
- Annotation model containing the semantic annotations of the CIM and PIM elements
- Mapping model describing predefined mappings between business components (CIM) and integratable elements or their interfaces (PIM)

The goal of the semantic conflict analysis is to perform semantical matching of the corresponding models, by establishing equivalent or compatible model elements using logical reasoning. The matching is performed at two levels: CIM-PIM, where business requirements (CIM) are matched against technical infrastructure (PIM), and PIM-PIM, where the semantic compatibility of the underlying systems which implement the integration scenario is investigated. The results of the semantic conflict analysis are mappings of data requirements to the underlying interface elements, mappings of required to provided interface elements and mappings of functional requirements to interfaces. The mappings thus mark the compatible (matched) interface elements. They are also used as a basis for the further conflict analysis phases. In case a semantic description of the models is not available or incomplete, the mappings are created automatically as far as possible and the remaining requirements have to be mapped manually. Figure 53 shows how artifacts such as models and annotations participate in the conflict analysis process as well as their relationships.

In the following sections the semantic conflict analysis algorithm is divided into several functional parts. Each part will be described by an activity diagram and additional explanation. The activity diagrams include actions and control flow to describe the process of the algorithm as well as objects nodes and object flow to define the produced or consumed artifacts of the algorithm. The

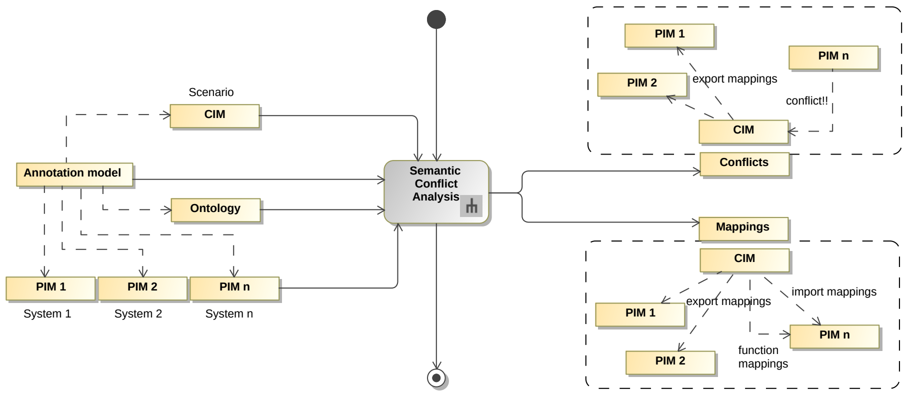


Figure 53: Model relations during semantic conflict analysis

algorithm description in this report is language and technology independent. Currently it is realized in Prolog, as described in Section 6.4.

6.1.1 Algorithm Overview

The semantic conflict analysis algorithm is described in a top-down approach, starting at an abstract level shown in Figure 54. The activities are refined and described in the subsequent subsections. At the top of the figure the model artifacts used during the algorithm are shown. For better readability of the diagram their object flows are not shown, because the are used in almost every activity.

The conflict analysis creates data-oriented and function-oriented requirements defined at the CIM level and then maps those requirements to the interface descriptions at the PIM level (requirement mappings). The CIM-PIM mapping model (MM) constraints the search space of interface descriptions. We differentiate between business object import and export requirements and business function requirements (definitions of the integration specialist) and import element requirements that occur during the conflict analysis of interface descriptions. All types of requirements are identified and mapped by the algorithm.

If all requirements are not fulfilled (mapped), the algorithm employs semantic reasoning to handle the unmapped requirements. Requirement identification and mapping are performed recursively to check whether newly generated mappings introduce additional requirements.

After recursion, data mappings are checked against functional mappings. The verification is necessary to eliminate possible wrong solutions in which interface parameters and elements match, but an interface provides a different functionality than required by a business function at the CIM level.

Furthermore an integration specialist has the option to manually map requirements that are still unfulfilled for business objects, functions and import elements. Furthermore, the algorithm identifies all possible requirement mappings. If multiple sources and targets are found, one can choose between them or keep the alternatives for further analysis phases.

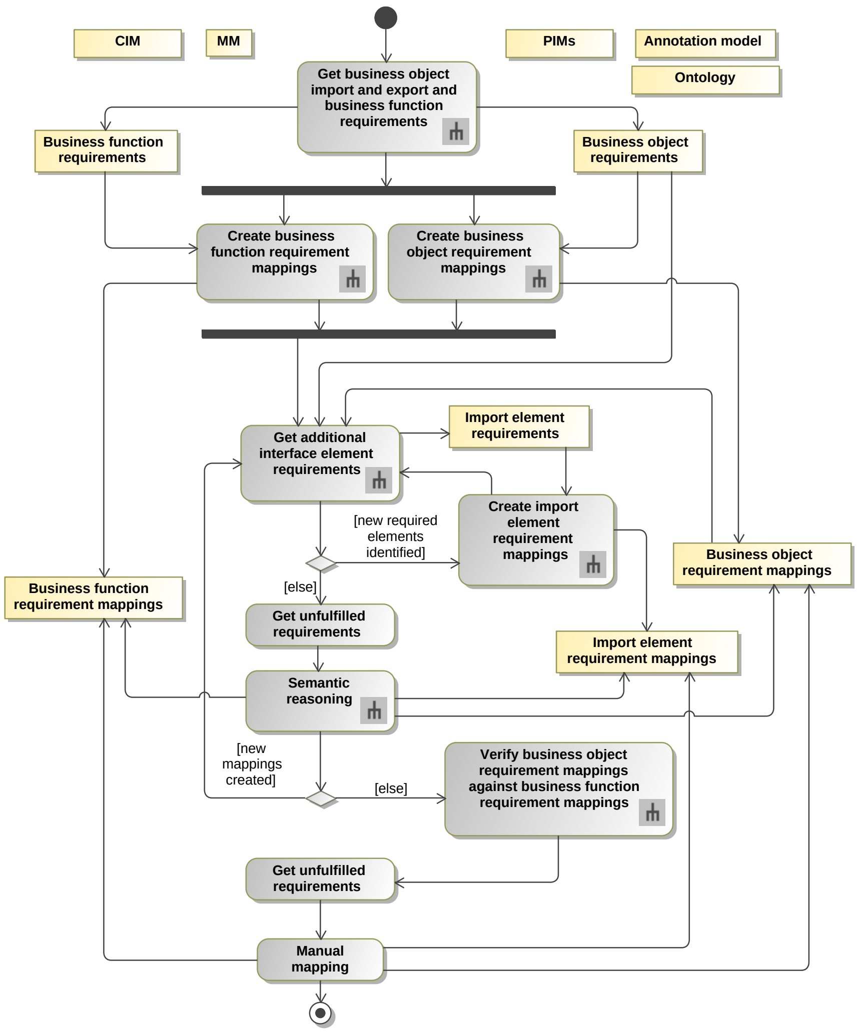

CIM
MM
PIMs
Annotation model
Ontology
Get business object import and export and business function requirements
Business function requirements
Business object requirements
Create business function requirement mappings
Create business object requirement mappings
Get additional interface element requirements
Import element requirements
Create import element requirement mappings
[new required elements identified]
[else]
Business function requirement mappings
Business object requirement mappings
Get unfulfilled requirements
Import element requirement mappings
Semantic reasoning
[new mappings created]
[else]
Verify business object requirement mappings against business function requirement mappings
Get unfulfilled requirements
Manual mapping

Figure 54: Semantic Conflict Analysis Overview

6.1.2 Get Business Object Import and Export and Business Function Requirements

Figure 55 shows refinement of the activity *Get business object import and export and business function requirements* in diagram 54. This activity creates the requirements according to the CIM business object and function definitions. All business objects and business functions of the integration scenario are iterated. If the business object's connection is linked to an export business interface, a business object export requirement is created. The same applies for import. Export and import requirements are not created if the connection is linked to a connector function instead of an interface. In this case the CIM explicitly defines that a certain functionality of the connector is used or generated, hence a matching interface of an existing system does not have to be identified. Business function requirements are created for each annotated business function. The function requirements specify that for each annotated business function an adequate implementation at the PSM/PIM level must be found. The requirements are used in the following activities to record mapping tasks and to avoid irrelevant searching while checking interface element dependencies.

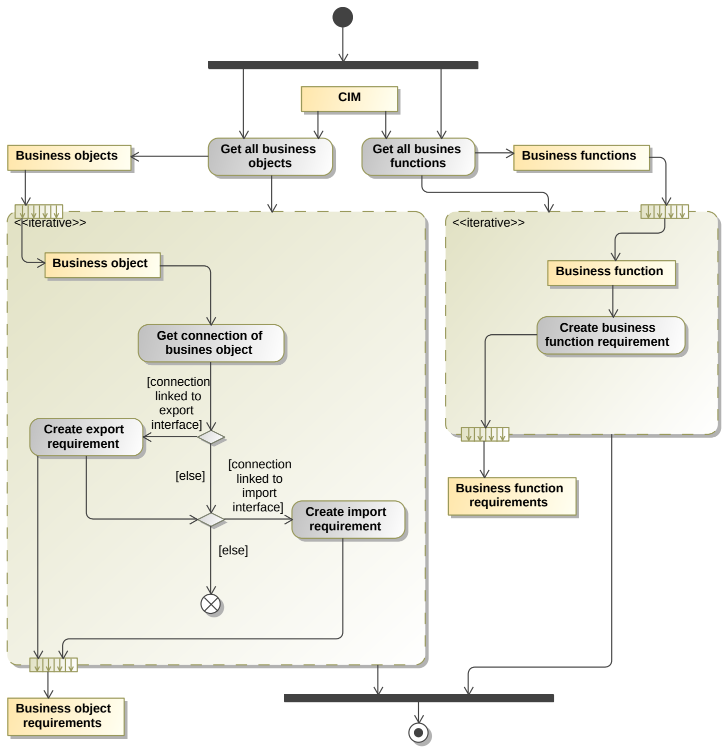

CIM
Business objects
Get all business objects
Get all busines functions
Business functions
<<iterative>>
Business object
Get connection of busines object
[connection linked to export interface]
Create export requirement
[else]
[connection linked to import interface]
Create import requirement
[else]
<<iterative>>
Business function
Create business function requirement
Business function requirements
Business object requirements

Figure 55: Generate business object import and export and business function requirements

6.1.3 Create Business Object Requirement Mappings

Figure 56 shows the refinement of the activity *Create business object requirement mappings* from diagram 54. The activity iterates over all previously created business object requirements and tries to find PIM elements that match the requirements using semantic annotations. Predefined mappings of business components to PIM interfaces or integratable elements in the mapping model (MM) are considered (if available) to avoid matching with superfluous underlying systems. The matched elements are iterated according to their usage in interfaces and complex types (context path) and a mapping is created for each context. Because of the type system at the PIM level, fields can have multiple occurrences when used in complex types of different parameters. To keep the diagram compact, export and import direction of requirements and PIM elements are not explicitly separated. If export requirements are processed, only exporting PIM elements are searched, likewise is done with import requirements and importing PIM elements. The result of the activity is a list of business object import and export requirement mappings.

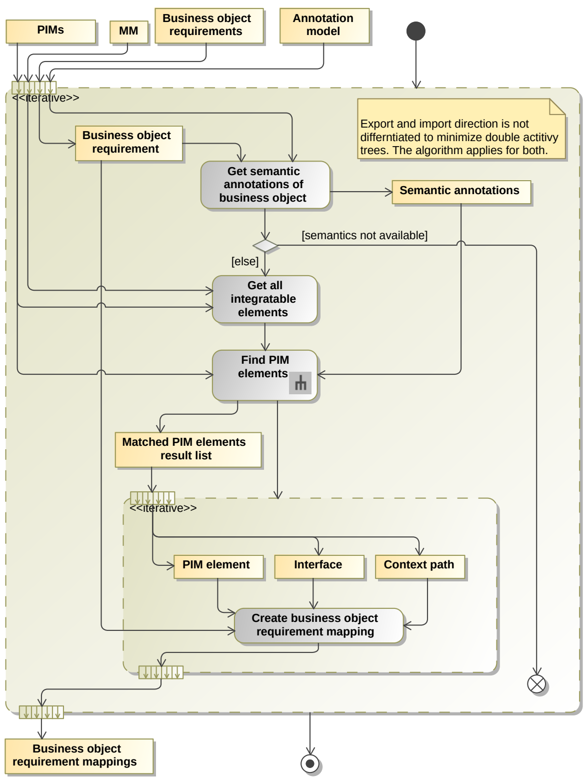

PIMs
MM
Business object requirements
Annotation model
<<iterative>>
Export and import direction is not differntiated to minimize double actitivy trees. The algorithm applies for both.
Business object requirement
Get semantic annotations of business object
Semantic annotations
[semantics not available]
[else]
Get all integratable elements
Find PIM elements
Matched PIM elements result list
<<iterative>>
PIM element
Interface
Context path
Create business object requirement mapping
Business object requirement mappings

Figure 56: Map business objects to export and import PIM elements

6.1.4 Create Business Function Requirement Mappings

Figure 57 shows the refinement of the activity *Create Business Function Requirement Mappings*. Business functions are mapped to technical PIM interfaces similar to the mapping of business objects. The algorithm considers all business function requirements and tries to find all possible matching PIM interfaces and combinations of them according to the functional semantic annotations. The result of the activity is a list of business function requirement mappings that fulfill the business function definitions at the CIM level. These mappings are used later to verify the usage of interfaces in business object mappings, that may have a structural match (accept the same parameters) but differ in functionality, that is, in how the parameters are processed.

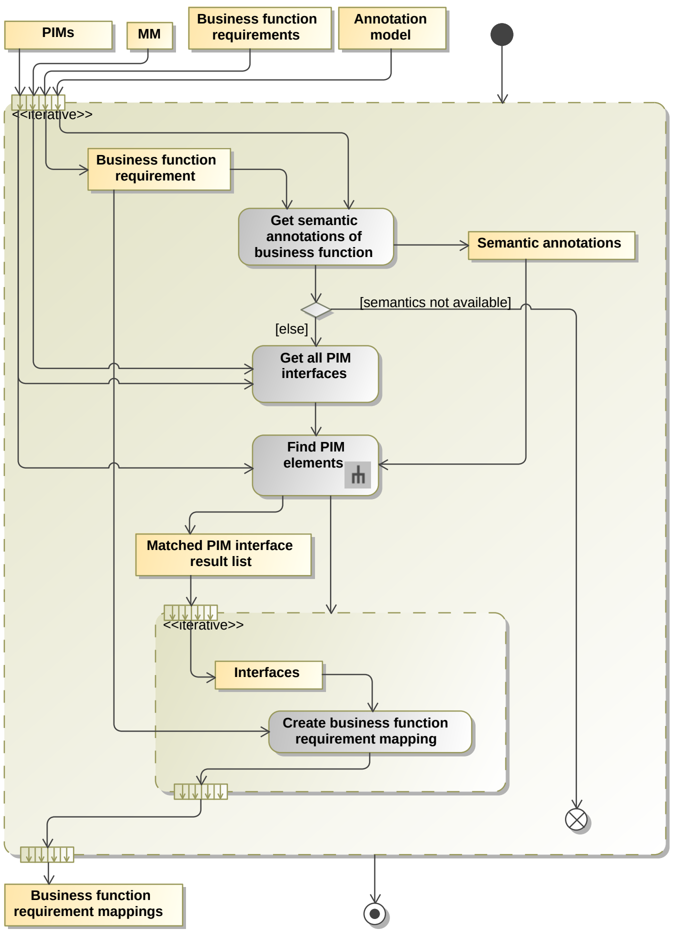
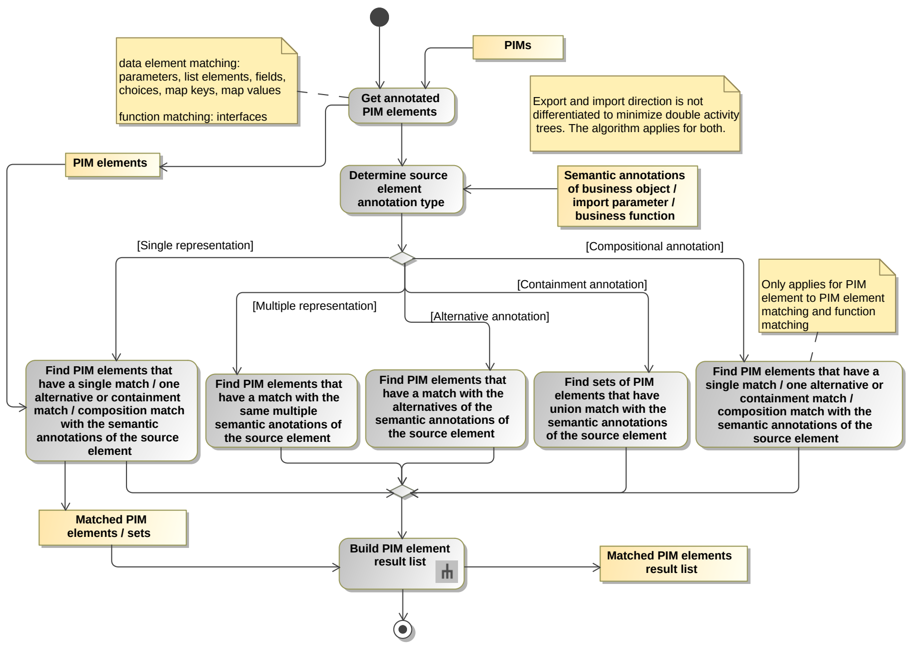

PIMs
MM
Business function requirements
Annotation model
<<iterative>>
Business function requirement
Get semantic annotations of business function
Semantic annotations
[semantics not available]
[else]
Get all PIM interfaces
Find PIM elements
Matched PIM interface result list
<<iterative>>
Interfaces
Create business function requirement mapping
Business function requirement mappings

Figure 57: Map business functions to PIM interfaces

6.1.5 Find PIM Elements

The *Find PIM elements* activity shown in figure 58 is used in three different parts of the algorithm. It generically describes how different model elements are matched according to their semantic annotations. First, the activity is invoked while creating the business object requirement mappings. In that context the source element of the comparison is a business object at the CIM level, whose semantic annotations are used to match (a set of) PIM data elements, such as parameters or fields of complex types. Secondly, importing PIM data elements are semantically matched against exporting PIM data elements while creating import element requirement mappings. Finally this activity is used by the create business function requirement mappings activity, in which semantic annotations of business functions at the CIM level are matched with interface annotations at the PIM level. The algorithm matches model elements according to the five annotation types: single representation, multiple representation, alternative, containment and compositional annotation (refer to section 5.4), split into separate activities.



PIMs
data element matching: parameters, list elements, fields, choices, map keys, map values
function matching: interfaces
Get annotated PIM elements
Export and import direction is not differentiated to minimize double activity trees. The algorithm applies for both.
PIM elements
Determine source element annotation type
Semantic annotations of business object / import parameter / business function
[Single representation]
[Compositional annotation]
[Containment annotation]
[Multiple representation]
[Alternative annotation]
Only applies for PIM element to PIM element matching and function matching
Find PIM elements that have a single match / one alternative or containment match / composition match with the semantic annotations of the source element
Find PIM elements that have a match with the same multiple semantic anotations of the source element
Find PIM elements that have a match with the alternatives of the semantic annotations of the source element
Find sets of PIM elements that have union match with the semantic annotations of the source element
Find PIM elements that have a single match / one alternative or containment match / composition match with the semantic annotations of the source element
Matched PIM elements / sets
Build PIM element result list
Matched PIM elements result list

Figure 58: Find PIM elements according to the semantic annotation

There are several possible combinations that result in a correct semantic match of model elements (CIM business object to exporting/importing PIM data elements; PIM importing elements to PIM exporting elements; CIM business functions to PIM interfaces). Figure 59 shows the annotation types in relation to each other. Each of the small images shows the source element(s) (filled squares at top) and the target PIM elements (white squares at bottom) in relation to concepts of the ontology (filled ellipses in the middle). Note that the compositional annotation is not applicable to business objects, therefore the last column is only valid for interface element matching and functional matching.

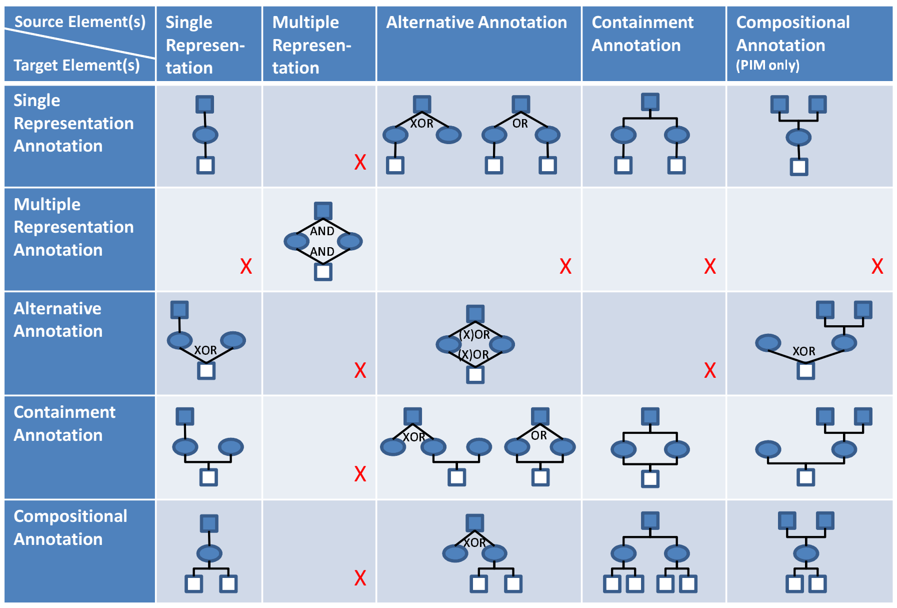


Figure 59: Model element matching depending on annotation types

All annotation types match with themselves. A single representation annotation matches with a part of the XOR-alternative annotation and with a part of the containment annotation. It also matches with compositional annotation, where multiple target elements compose the same ontology concept. The multiple representation annotation does not match with other types except itself. The alternative annotation with the *OR*-operator matches with a set of single annotations and the containment annotation. *XOR*-annotations match with all types except multiple annotation, because one of the alternatives is sufficient.

We will describe in more detail the matching of the containment annotation as source element annotation (fifth column in Figure 59) with the other types, in the context of processing a business object export requirement. This kind of requirement means that the integration process requires the export of certain data, represented by the business object at the CIM level. The containment annotation is used if the business object's level of detail is coarse and the concepts of the ontology are more detailed (imagine `Customer data` annotated with concepts *Address, Bank Account*). If such an annotation is processed, one or more matching interface elements at the PIM level should be located, based on their annotations. A direct match with an equal containment annotation is obvious. Alternative annotation does not directly match, because containment demands the presence of all single elements together, neither does the multiple representation annotation, because this type implies polymorphy. Single representation matches containment if a set of exporting PIM interface elements can be found that has a union match (e. g., one interface parameter annotated with *Address* and a second with *Bank Account*). Compositional annotation matches containment if multiple PIM elements are annotated together with an equal set of ontology concepts (e. g., interface parameters `Name`, `Street`, `Town` together annotated with *Address*, and another parameter set of `Name`, `AccountID` and `BankID` annotated with *Bank Account*). Mixed combinations are possible, e. g, one single representation annotation and one part of an XOR-alternative annotation. Finding such sets of interface elements will result in a mapping that fulfills the requirement defined by the business object.

Compositional annotation links multiple source PIM elements to a single concept of the ontology. Therefore the search strategy for target elements conforms to the matching of single annotations. During the iteration of requirements it can happen that multiple source elements are mapped, of which some have not yet been iterated. The algorithm verifies whether it processes such elements.

The empty cells in Figure 59 marked with X show that a direct match between the two annotation types is not possible. Additionally mixed combinations are possible as described previously (not illustrated in the table).

6.1.6 Build PIM Element Result List

After having identified PIM elements, the results have to be checked regarding their usage in different interfaces. The PIMM offers metaclasses to describe a type system in which the structural model elements can be reused among different types and parameters (see section 1.1.4). Matched parameters are unique, but e. g., matched fields of complex types can have multiple occurrences, if the complex type is used for different parameters or even again for fields. Figure 60 illustrates the algorithm for iterating all matched PIM elements to determine their multiple usage. The activity does not apply to matched interfaces for business functions.

Suppose that a structural model at the PIM level defines a complex type *CustomerName* that consists of the fields *firstname* and *lastname*. That type is once used for an export parameter *customer* of interface *getCustomer* as well as for import parameter *cust* of interface *saveCustomer*. If the field *lastname* is matched by the algorithm, the result leads to two possibilities of requirement mappings (one export and one import mapping). The context paths for this example are:

- `getCustomer.customer.CustomerName.lastname`
- `saveCustomer.cust.CustomerName.lastname`

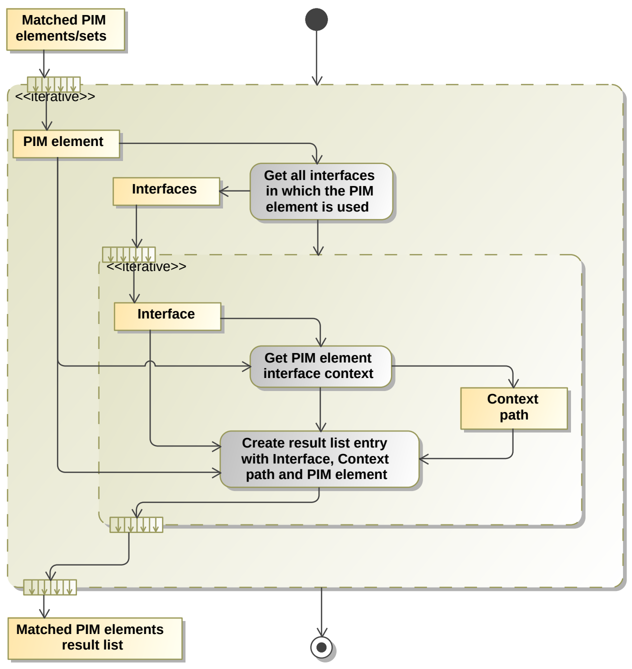

Matched PIM elements/sets
<<iterative>>
PIM element
Get all interfaces in which the PIM element is used
Interfaces
<<iterative>>
Interface
Get PIM element interface context
Context path
Create result list entry with Interface, Context path and PIM element
Matched PIM elements result list

Figure 60: Build PIM element result list of matched PIM elements

6.1.7 Get Additional Interface Element Requirements

Figure 61 shows the *Get Additional Interface Element Requirements* activity refinement, that was defined in diagram 54. The activity determines additional requirements for import elements that have to be passed to an interface. As explained in the previous sections, business object requirement mappings have been created which refer to a certain set of interfaces. These interfaces are taken into account for the further analysis of import elements (parameters, fields, etc.). Import element requirements are checked recursively, because generated requirement mappings can introduce new interface dependencies. If the algorithm finds elements that are already mapped or required, a new requirement is not created to determine the end of the recursion and to avoid interface cycles. As described in the previous section, import elements of complex types can be reused. Requirements are created for each use.

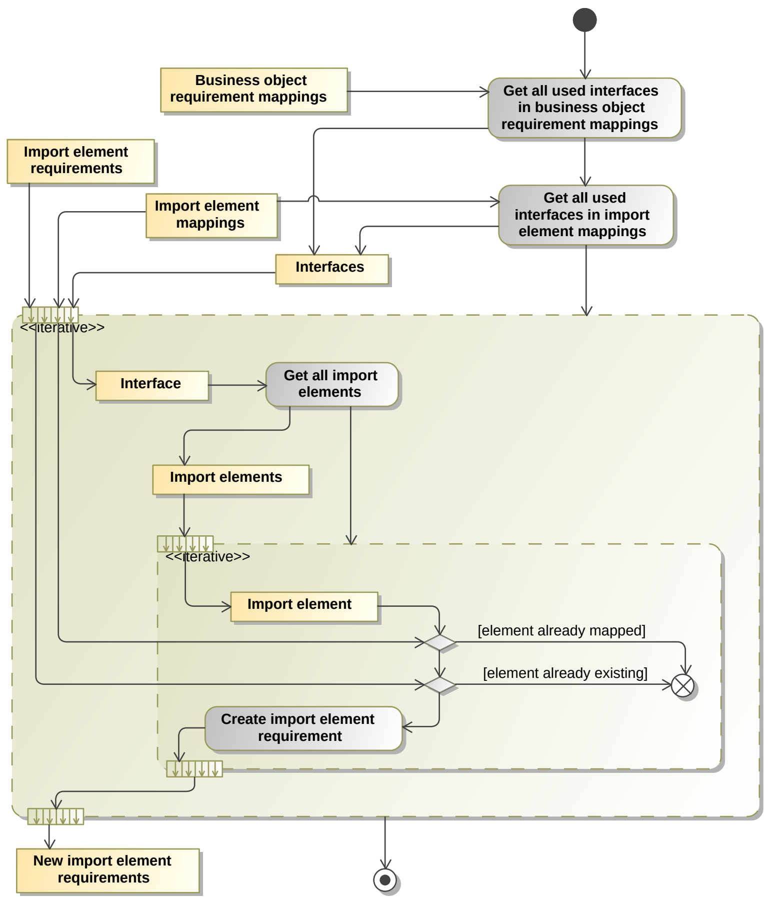

Business object requirement mappings
Get all used interfaces in business object requirement mappings
Import element requirements
Import element mappings
Get all used interfaces in import element mappings
Interfaces
<<iterative>>
Interface
Get all import elements
Import elements
<<iterative>>
Import element
[element already mapped]
[element already existing]
Create import element requirement
New import element requirements

Figure 61: Generate requirements for import elements

6.1.8 Create Import Element Requirement Mappings

The refinement of the *CCreate Import Element Requirement Mappings* activity shown in figure 62 iterates the previously created import element requirements and searches for exporting PIM elements that semantically match. Multiple export possibilities are considered (also according to the usage of PIM elements in different interfaces) and a mapping is created for each of them. This activity makes use of the *Find PIM elements* described in section 6.1.5, because the matching of business objects to PIM elements is similar to the PIM element matching (except for the explicit direction and use of compositional annotations). The algorithm searches in all available interfaces descriptions, that may optionally be constrained by the CIM-PIM mapping model. If an exporting element is found, and the interface has not yet been analyzed, it will be queued for further analysis of parameter dependencies.

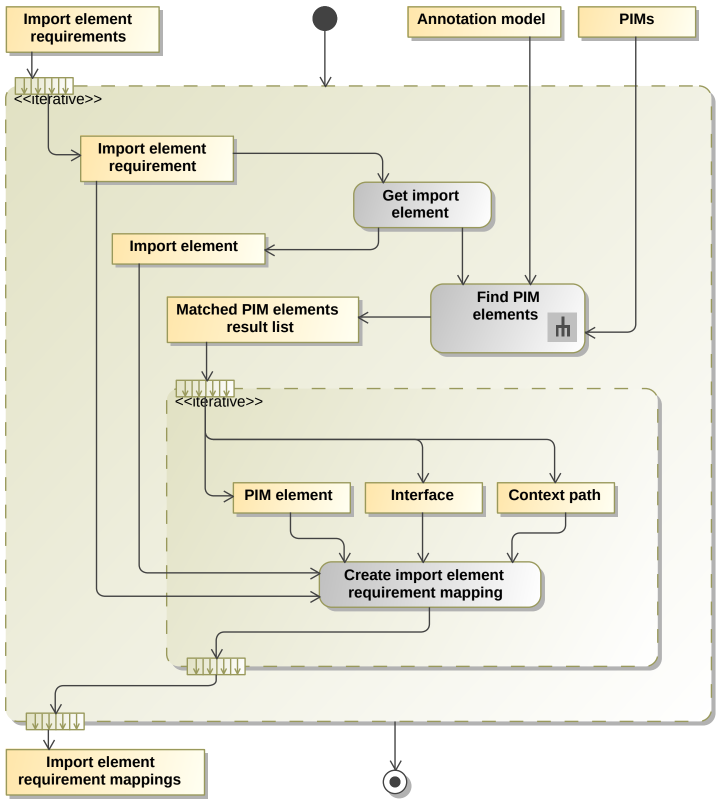

Import element requirements
Annotation model
PIMs
<<iterative>>
Import element requirement
Get import element
Import element
Find PIM elements
Matched PIM elements result list
<<iterative>>
PIM element
Interface
Context path
Create import element requirement mapping
Import element requirement mappings

Figure 62: Create mappings for import elements depending on exporting PIM elements

6.1.9 Semantic Reasoning

After the iteration of the five major analysis steps for business objects, import elements and business functions (see diagram 54), requirements and dependencies may still not be satisfied. *Semantic reasoning* will be used, if n:m matching of annotations is not sufficient. The semantic reasoning part of the algorithm describes how additional mappings can be identified by reasoning over the ontology, the CIM/PIM and annotation models. Figure 63 shows the general procedure of the iteration of unfulfilled requirements, the different reasoning strategies are described in Section 6.1.10.

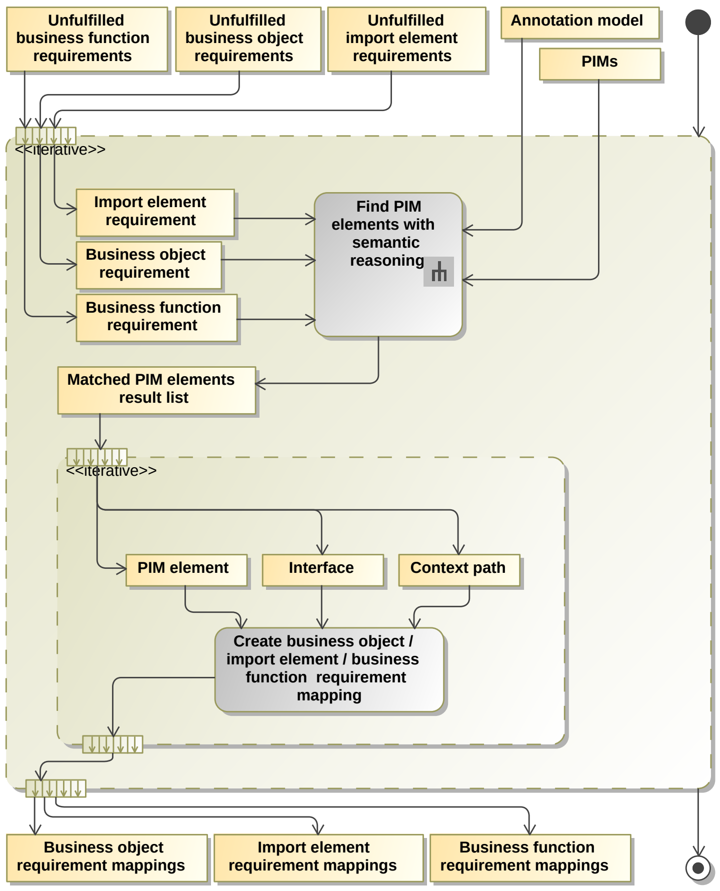

Unfulfilled business function requirements
Unfulfilled business object requirements
Unfulfilled import element requirements
Annotation model
PIMs
<<iterative>>
Import element requirement
Business object requirement
Business function requirement
Find PIM elements with semantic reasoning
Matched PIM elements result list
<<iterative>>
PIM element
Interface
Context path
Create business object / import element / business function requirement mapping
Business object requirement mappings
Import element requirement mappings
Business function requirement mappings

Figure 63: General procedure of semantic reasoning for unfulfilled requirements

6.1.10 Find PIM Elements with Semantic Reasoning

Figure 64 shows the activity *Find PIM elements with semantic reasoning*, that includes the strategies to identify interfaces and interface elements for unfulfilled requirements. All relevant models are transformed into facts of a particular reasoner language. Afterwards the reasoning engine applies the rules illustrated in the figure. The results are transformed back to our model infrastructure and then are added to the previously created mappings. After semantic reasoning, the recursion follows back to the activity *Get additional interface element requirements* (see figure 54 on page 65) to analyze additional found interfaces.

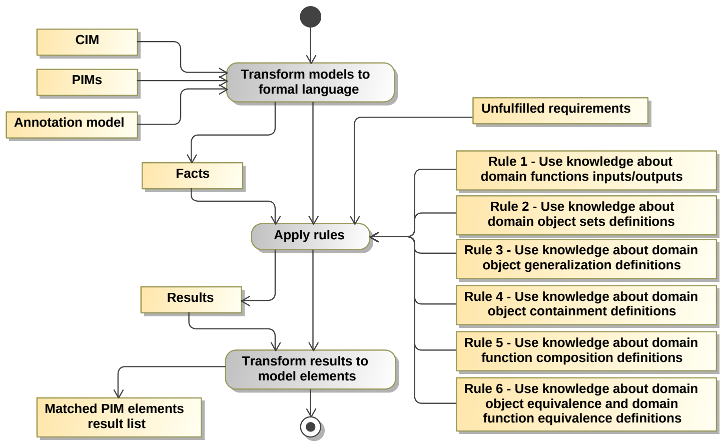

CIM
PIMs
Annotation model
Transform models to formal language
Unfulfilled requirements
Facts
Rule 1 - Use knowledge about domain functions inputs/outputs
Rule 2 - Use knowledge about domain object sets definitions
Apply rules
Rule 3 - Use knowledge about domain object generalization definitions
Results
Rule 4 - Use knowledge about domain object containment definitions
Rule 5 - Use knowledge about domain function composition definitions
Transform results to model elements
Rule 6 - Use knowledge about domain object equivalence and domain function equivalence definitions
Matched PIM elements result list

Figure 64: Strategies of reasoning to resolve semantic conflicts

We defined the following rules that utilize the knowledge of defined predicates in the ontology. Rule 1 evaluates `Input` and `Output` of domain functions to locate implementations that can be used for data transformation. Consider an interface with import parameter A annotated with the concept α and a second interface with export parameter B annotated with the concept β. All domain functions F_i and their respective implementations are discovered, that have α as output and β as input. If an implementation is not available, a transformer connector function stub is generated.

Rule 2 investigates the ontology knowledge of `ListOf`-predicates to find interface elements that deliver data sets of which single elements can be extracted. Imagine import parameter A annotated with the concept α. The rule finds all exporting interface elements B_i annotated with β_i, where β_i `ListOf` α. The result requires connector functions, that iterate data sets and call interfaces consecutively.

Rule 3 uses generalization definitions (`IsA`) in the ontology. Consider import parameter A annotated with the concept α. The rule matches all exporting interface elements B_i annotated with β_i, where β_i is a specialization of α. The result requires a connector function, that transforms the objects of import parameter A into the representation of B_i by filtering out all specialized information of B_i.

Rule 4 evaluates the containment definitions in the ontology, that are expressed between domain objects by `has`-predicates to find composable parts of unmatched interface elements. If an import parameter A annotated with concept α cannot be satisfied, the rule processes all set of concepts $\beta_1 ... \beta_n$ that are referenced with the `has`-predicates. If for each β_i an exporting interface element B_i can be found, A is composed of B_i. The results require aggregating connector functions.

Rule 5 is the equivalent to rule 4, except that it uses functional instead of object composition. Domain functions and their implementations are composed to fulfill a business function requirement. Functional composition must be further validated by the behavior conflict analysis to comply with pre- and post-conditions.

Finally, rule 6 takes equivalence information into account, that is explicitly defined by `IsEquivalentTo`-predicates in the ontology. Consider business function A annotated with domain function α. The rule identifies all domain functions β_i, where β_i `IsEquivalentTo` α.

6.1.11 Verify Business Object Requirement Mappings against Business Function Requirement Mappings

Before the integration specialist has the option to manually map unfulfilled requirements, the semantic conflict analysis cross-checks all created business object requirement mappings with business function requirement mappings. The additional verification is necessary because a semantic mapping based on data-oriented information only could lead to false positive matches. Identified interface elements can match, but the provided or required interface implements different functionality than specified at the CIM level. Refer back to scenario 7 (section 3.12) in which `getPhoneLocation` and `getSupportLocation` have the same import and export parameters, the same input/output data semantics but different functionality.

Figure 65 shows the verification process. It is based on comparing interfaces, that are referenced by both types of requirement mappings. On the one hand, interfaces are determined through the business function requirement mappings. Business functions are identified that produce or consume business objects and functional mappings are searched for them. The result is *BF Mapping Interfaces*. On the other hand, PIM elements of the business object requirement mapping are investigated. Each of them belongs to an interface, resulting in *BO Mapping Interfaces*. The interfaces on each side are compared and if they don't match, the business object requirement mapping is discarded.

In case of pure data import or export business process the data mappings cannot be functionally verified, because business objects are not necessarily associated with business functions. The verification is also not applicable if functional mappings are not available due to missing functional annotation or non-solvable functional conflicts.

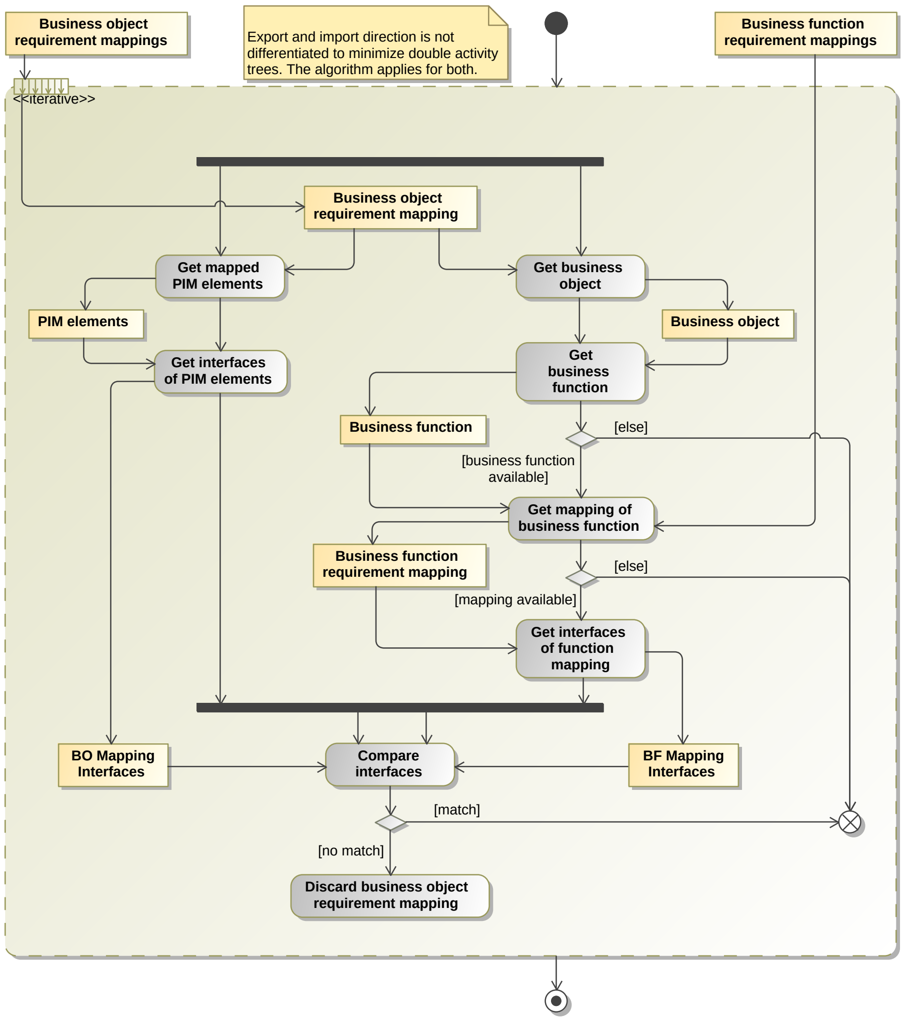

Business object requirement mappings
Export and import direction is not differentiated to minimize double activity trees. The algorithm applies for both.
Business function requirement mappings
<<iterative>>
Business object requirement mapping
Get mapped PIM elements
Get business object
PIM elements
Business object
Get interfaces of PIM elements
Get business function
Business function
[else]
[business function available]
Get mapping of business function
Business function requirement mapping
[else]
[mapping available]
Get interfaces of function mapping
BO Mapping Interfaces
Compare interfaces
BF Mapping Interfaces
[match]
[no match]
Discard business object requirement mapping

Figure 65: Verification of requirement mappings

6.2 Conflict Analysis Results Representation

In this section we describe the Conflict Analysis Metamodel (CAMM), which is used to represent detected requirements and created requirement mappings during the conflict analysis process. Examples of its use are given in section 6.3. The semantic conflict analysis algorithm generates requirements while analyzing integration scenarios at the CIM level and interface descriptions at the PIM level (see section 6.1.3, 6.1.4 and 6.1.7). Figure 66 shows the four possible types of requirements related to the appropriate metaclasses of the CIMM and PIMM. `BusinessObjectRequirement` references the `BusinessObject` metaclass of the CIMM. Depending on the transfer direction the subclasses are either linked to `ImportInterface` or `ExportInterface`. `BusinessFunctionRequirement` is related to the CIMM's `BusinessFunction`. The metaclass `ImportElementRequirement` is used to mark an importing interface parameter or part of its structure. The `related`-association refers to the `AnnotatableElement` metaclass, which is base class for e. g., `FunctionParameter`, `Field`, `ListElement`. The association is constrained to PIM elements, that are in an importing role (e. g., a certain `Field`, that is part of a complex type, which is used as an `ImportParameter`).

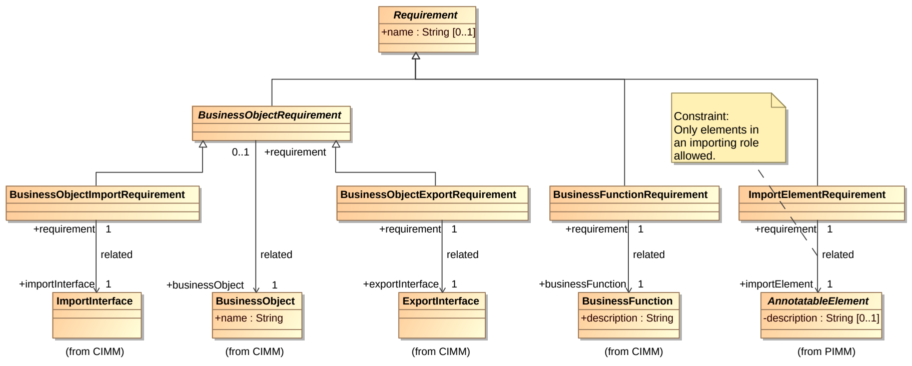


Figure 66: CAMM – Requirement metaclasses related to CIMM and PIMM

In an analogous manner to the requirements, the CAMM contains four requirement mapping metaclasses, that are instantiated to create CIM-PIM and PIM-PIM mappings: the abstract `BusinessObjectRequirementMapping` with specializations `BusinessObjectImportRequirementMapping` and `BusinessObjectExportRequirementMapping`, as well as `BusinessFunctionRequirementMapping` and `ImportElementRequirementMapping`. The requirement mappings will be defined in the following diagrams.

Figure 67 depicts the metaclass `BusinessObjectRequirementMapping`, that is used to map business object requirements to either exporting or importing interface elements at the PIM level. A mapping satisfies exactly one requirement, and a requirement can be satisfied by several mappings, in case the algorithm finds multiple solutions for a requirement. All PIM elements that satisfy a requirement are referenced with the `target`-association linked to the `AnnotatableElement` (e. g., parameters, fields). The association is constrained by the mapping's type. In case a `BusinessObjectImportRequirementMapping` is used, only elements with an importing role may be referenced. The association to the `Interface` of the PIMM holds the information to which interface the discovered elements belong.

The `ElementContext` must be used if the target elements are parts of complex data types, because it is possible to reuse types among several parameters or fields. The `interface`-association, the `ElementContext` and the `target`-association is a model representation of the dot notation (e. g., *getMostValuableCustomer.customer.firstname*) introduced in the motivation scenario descriptions in chapter 3. An instance of the context references the path between the interface and the target elements as an alternating sequence of `TypedElement` and `Type` instances at the PIM level. The sequence of the context must begin with a typed element (e. g., a parameter) and must end with a complex type.

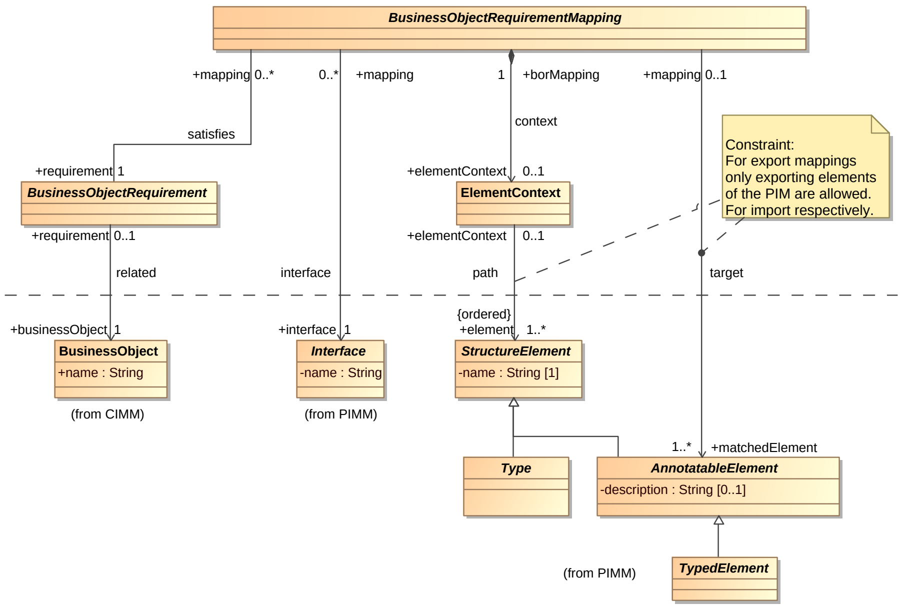


Figure 67: CAMM – Business Object Requirement Mappings

The second type of mapping, the `BusinessFunctionRequirementMapping`, is instantiated during the analysis of functional requirements defined at the CIM level (see section 6.1.4). Figure 68 illustrates this metaclass and its relation to the functional metaclasses of the CIMM and PIMM. The mapping establishes the link between the business function requirements and implementations described by `Interfaces` at the PIM level. A requirement can be satisfied by several mappings, if multiple functional equivalent interfaces are found.

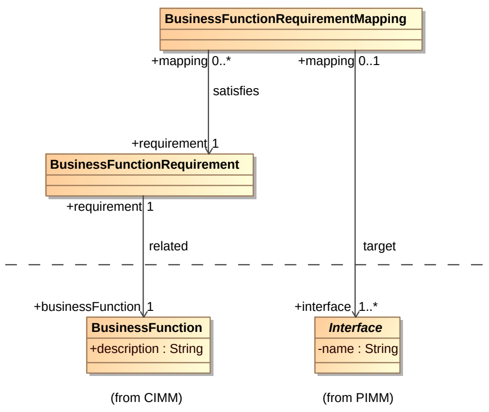


Figure 68: CAMM – Business Function Requirement Mapping

In Figure 69 the `ImportElementRequirementMapping` is shown. Requirements for import elements are generated during the analysis of interfaces at the PIM level that have been identified by the analysis of business object requirements. The mapping links import element requirements to exporting interface elements on PIM level. The definition of the metaclass is similar to the business object requirement mapping, except for the `related`-association to the `ImportElementRequirement` and the constraint that only exporting interface elements can be referenced with the `target`-association. The element context is used in the same way as for business object requirement mappings.

Finally the CAMM provides `ConnectorMapping` metaclasses (Figure 70) in order to describe analysis results that require further processing by connector functions. In section 6.1.10 we defined six rules for reasoning over the ontology and CIM/PIM models to solve conflicts of interface mismatches. The results of the rules are described with three types of connector mappings: `AggregationMapping` (n:1 processing), `TransformationMapping` (n:m processing) and `SplitMapping` (1:n processing). These types of mappings correspond to a subset of connector functions at the CIM level (see section 1.1.1). Each of the mappings references the `ImportElementRequirementMapping` via a `consume`- and `produce`-association, whose cardinalities depend on the mapping type.

6.3 Conflict Analysis Results Example

After having described the metamodel for conflict analysis results, we have selected three of the motivating scenarios to demonstrate the features of the CAMM. The results of Scenario 1 show the simplest case of business object and import element requirement mappings. With Scenario 2 we demonstrate how the element context is applied. Scenario 3 describes a more complex analysis result, in which a connector function is needed to aggregate interface elements for a target system.

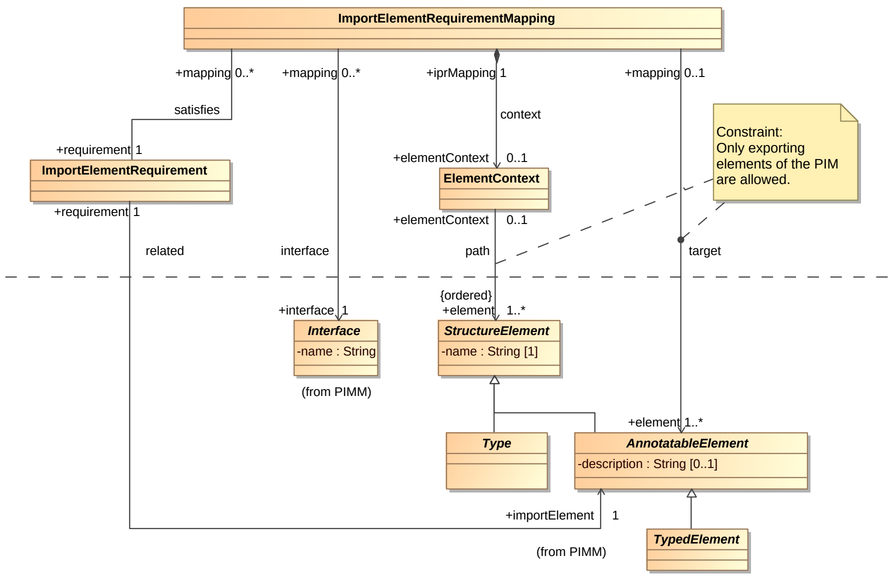


Figure 69: CAMM – Import Element Requirement Mapping

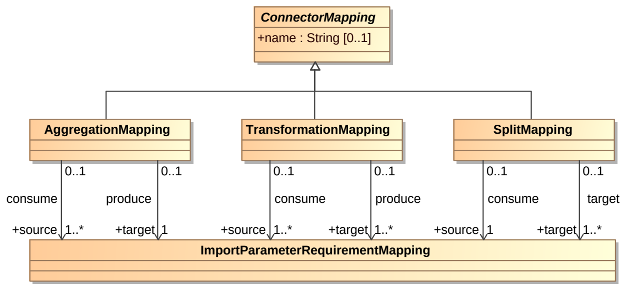


Figure 70: CAMM – Connector Mappings

6.3.1 Scenario 1 results: Simple requirement mappings

Motivating scenario 1 (section 3.1) described the transfer of a simple business object *Sum of sales* at the CIM level. The underlying systems' PIM interfaces have one export parameter *sum* and one import parameter *turnover*, respectively. Parts of the CIM are shown at top of Figure 71, PIMs are displayed at bottom. In-between the conflict analysis model is shown. It contains three requirements, that have been generated for the business object (export and import), and for the import parameter. All requirements are satisfied with a corresponding mapping, linking the CIM business object to the PIM export and import parameter, and linking the ERP's import parameter to the Webshop's export parameter.

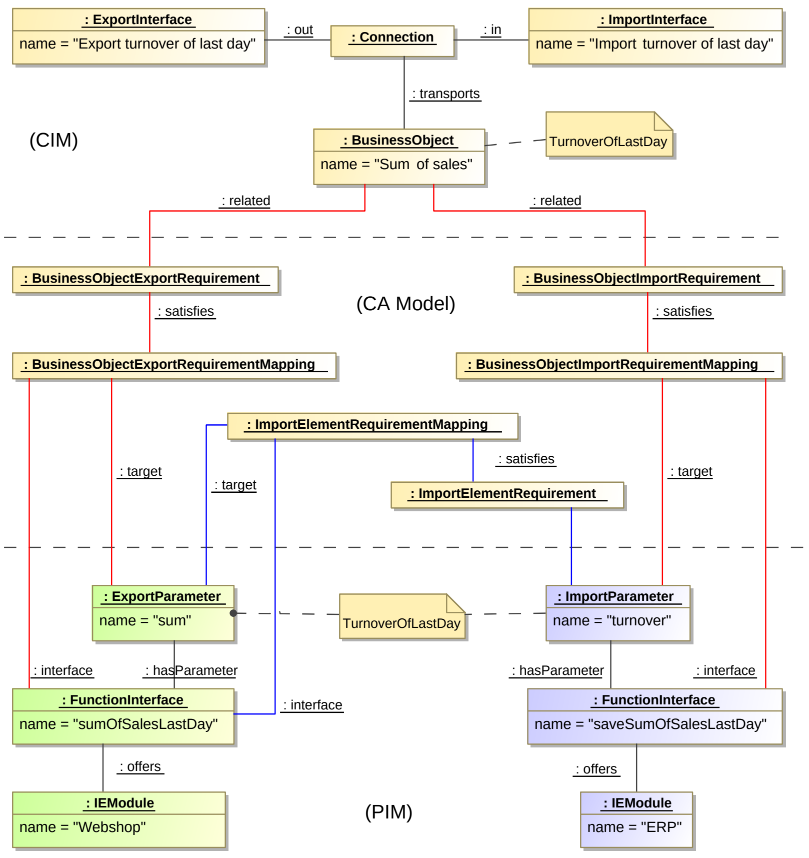


Figure 71: Conflict analysis results for scenario 1

6.3.2 Scenario 2 results: Interface element context

In section 3.3 a more complex scenario was defined, that requires the transfer of structured data. In Figure 72 the business object structure is shown. *Customer ID* will be exemplary selected to illustrate requirement mapping to a substructure part of interface descriptions. The export mapping references the function interface *getCustomerListForLastDay* with the `interface`-association. The matching PIM element is the field *id*, referenced with the `target`-association. Because the types *customerType* and *cuListType* can be used by any other parameter or field of the model, even as import, an element context is created, that references the path between the interface and the target (the order of the sequence cannot be seen in the figure due to limitations of the diagram type). The mapping (interface, context, target) is equivalent to the expression *getCustomerListForLastDay.customers.cuListType.customer.customerType.id*.

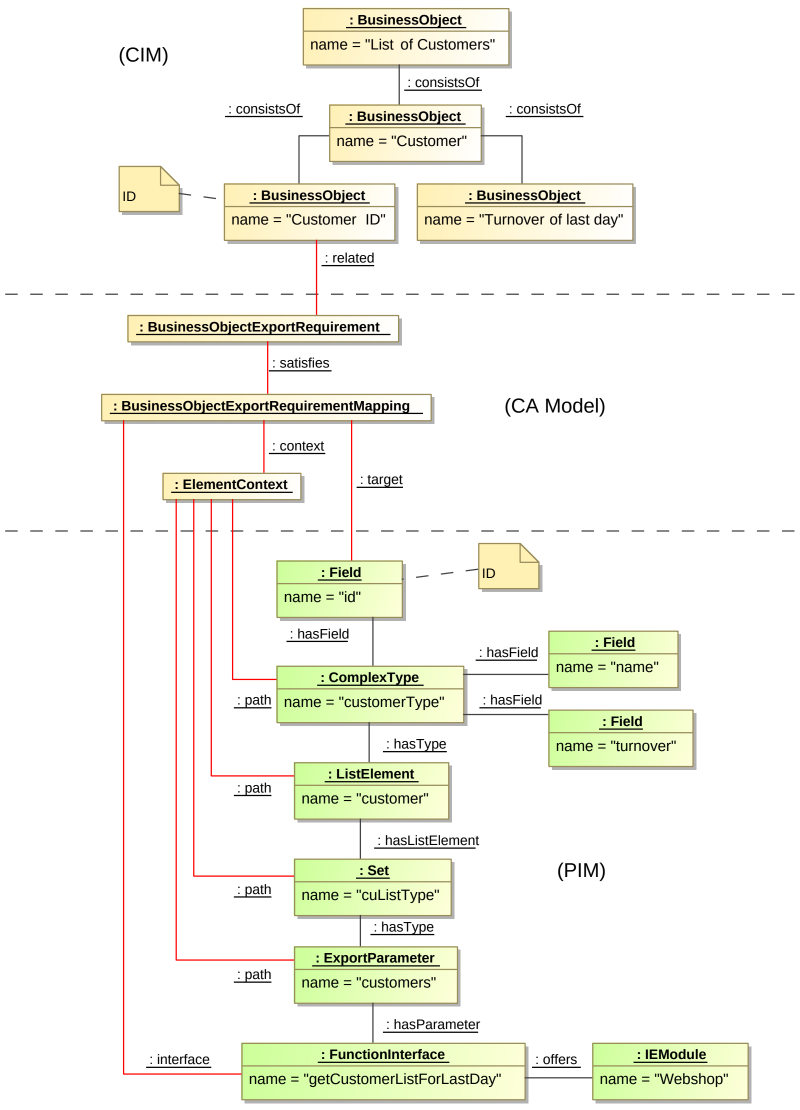


Figure 72: Conflict analysis results for scenario 2 (excerpt)

6.3.3 Scenario 3 results: Connector mapping

Scenario 3 contains a conflict in which the business object definition at the CIM level cannot be directly met by the underlying systems (section 3.8). The semantic conflict analysis is able to fulfill the import requirement for business object *Name* by mapping to import parameter *name* of the ERP system (Figure 73). Through semantic reasoning (see rule 4 in section 6.1.10) the algorithm resolves the conflict with the result, that *Name* can be composed by the fields *firstname* and *lastname* of the Webshop's complex type. The rule implies the creation of an aggregating connector function to merge both fields for the import parameter.

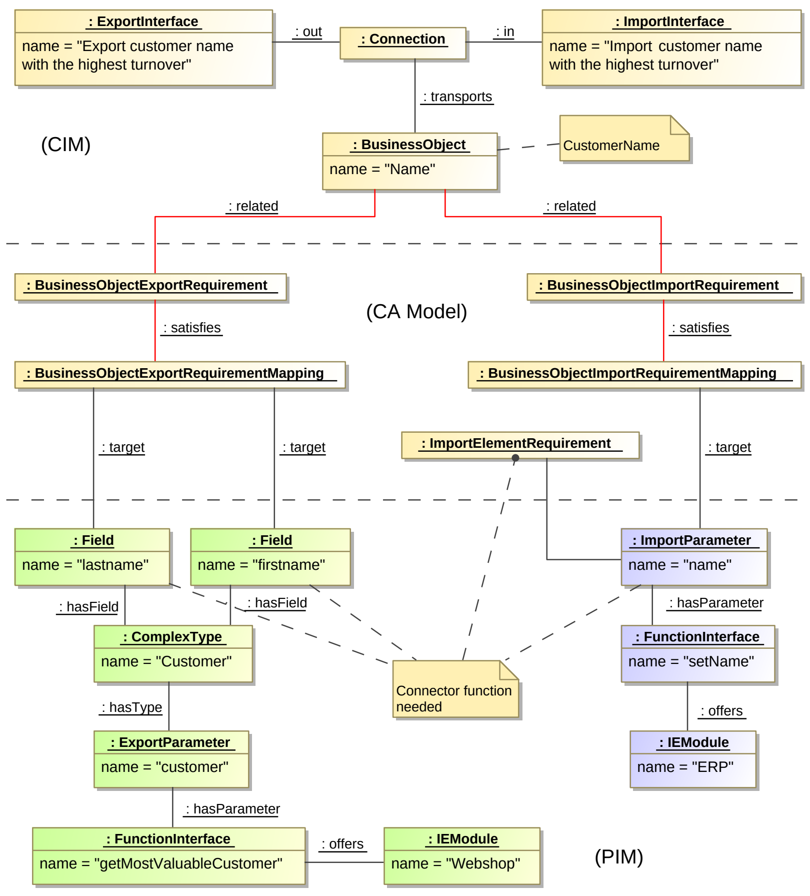


Figure 73: Conflict analysis results for scenario 3 – part 1

The information about the required connector function must be kept for further connector model generation. Three steps are necessary to create the adequate mappings: first, an interface description at the PIM level for the connector function is generated, in case it is not yet available (e. g., as domain function). Second, the usual import element requirement mappings that match the connector function are created (Figure 74). Third, an `AggregationMapping` is created, that references the mappings of *firstname* and *lastname* with the `consume`-association and the mapping of *name* with the `produce`-association.

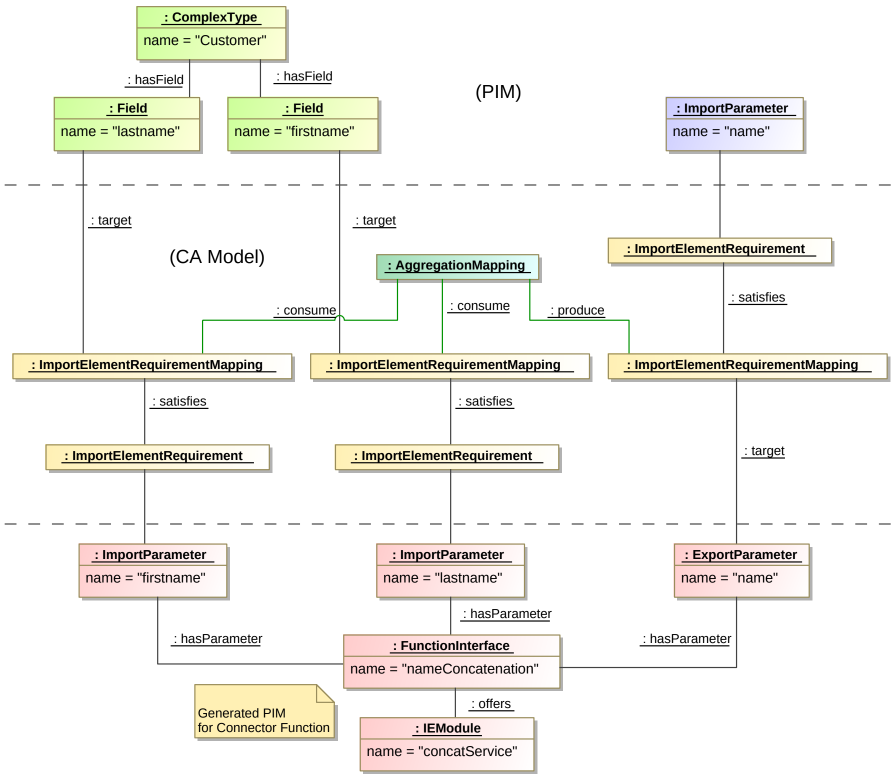


Figure 74: Conflict analysis results for scenario 3 – part 2

6.4 Semantic Conflict Analysis Implementation

Semantic conflict analysis is implemented as part of the Model-Based Integration Framework (MBIF). The conflict analysis algorithm is encapsulated in an Eclipse plug-in which uses the Prolog engine for semantic reasoning and analysis of components & integration requirements. The implementing approach is sketched in Fig. 75

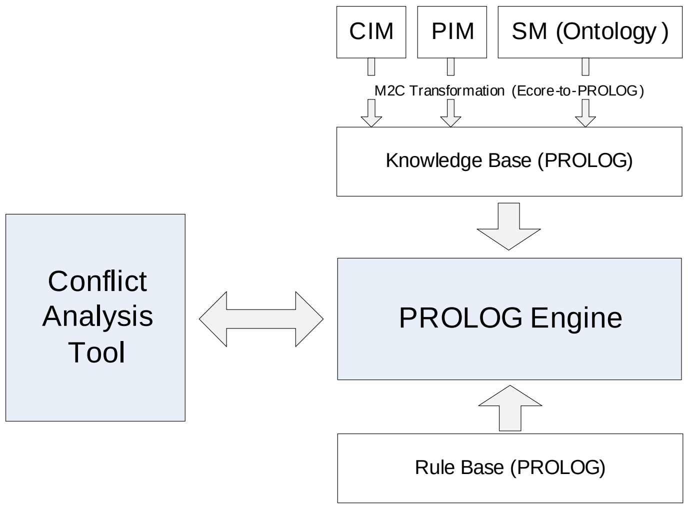


Figure 75: Semantic Conflict Analysis implementation

To enable the semantic conflict analysis, model artifacts are transformed to Prolog and form the knowledge base for the semantic reasoning. Parts of the semantic conflict analysis algorithm introduced in 6.1 as well as the structure of CIMM, PIMM, and SMM are implemented in Prolog and constitute the corresponding rule base. The constituent parts of the implementation are explained in the following sections.

6.4.1 Prolog Engine

The SWI-Prolog environment is used as Prolog engine. The SWI-Prolog provides a library (JPL) which enables to embed Prolog in Java and Java applications in Prolog. In our approach, the JPL's Java API is used by the conflict analysis tool to access the SWI-Prolog engine.

6.4.2 Ecore-to-Prolog Transformation

The model artifacts required for the semantic conflict analysis and represented as Ecore models (CIM, PIM, SMM) are parsed by the conflict analysis tool. The model elements are identified by the unique identifiers generated by the EMF framework when creating the model artifact. According to the artifact's metamodel for each model element (class, attribute or relation) the corresponding Prolog fact (predicate) is generated. The predicate name comprises the metamodel type and the name of the appropriate model element (including the metamodel package name). Classes are represented as unary predicates containing the class name in the predicate name and the element identifier as predicate's parameter, e. g.,

```
pimm_str_Field(FieldId)
```

represents an instance of the metaclass `Field` in the PIMM package `structure`, where *FieldId* is the given element identifier of the class instance. The relations between the classes are represented by the binary predicates named according to the name of the appropriate relation in the metamodel, e. g.,

```
pimm_Offers(IEModulId, FunctionInterfaceId)
```

describes the relationship `offers` between instances of the PIMM classes `IEModul` and `FunctionInterface`. The class attributes are represented as ternary predicates in the form of

```
attribute(ElementID, AttrName, AttrValue)
```

where *ElementId* is the identifier of the corresponding parent element (class), *AttrName* and *AttrValue* is the name and value of the given attribute.

6.4.3 Knowledge Base & Rule Base

The semantic conflict analysis implementation is based on the data provided by the knowledge base and the rule base. The knowledge base contains the Prolog representation of participating model artifacts: SM, CIM, PIM. The partial implementation of the semantic conflict analysis algorithm as well as rules describing generalization relationships in the appropriate metamodels are contained in the rule base. An example of such generalization relationship is the relationship between the classes `FunctionInterface`, `DocumentInterface`, `MethodInterface`, and `Interface`:

```
pimm_Interface(I) :- pimm_FunctionInterface(I) |
pimm_DocumentInterface(I) | pimm_MethodInterface(I).
```

For each integration scenario the same rule base is used as long as the metamodels (CIMM, SMM, PIMM etc.) remain unchanged. The knowledge base is dependent on the integration scenario and models participating in it.

The following exemplary rule (simplified) implements a part of the conflict analysis algorithm to determine the possibility of extraction of given business object from the set of available interfaces. The business object annotated with the semantic concept can be exported if at least one of the interfaces delivers an output element annotated with the equivalent semantic concept.

```
exportableBO(BO):-smm_BO(BO),((smm_AnnotatedWith
(BO, SemanticConcept), isAvailableForExport
(SemanticConcept))|(isComplexBO(BO), forall
(cimm_PartOfBO(PartBO, CompoundBO),exportableBO(PartBO)))).
```

The rule makes use of `isAvailableForExport` and `isComplexBO` rules.

7 Summary

This document describes the semantic conflict analysis framework developed in the course of the BIZYCLE Project. The framework is based on the Model-Driven Architecture (MDA) in order to extend the limitations of the existing semantic annotation and reasoning frameworks towards many unsupported platforms. The goal was to build an universal MDA framework which can be easily extended to support any kind of system data and/or interfaces under the single semantic (ontology) and annotation metamodel.

In the Chapter 1, the BIZYCLE project was introduced and relevant metamodels explained. Chapter 2 covered related work in the area of semantic annotation and reasoning. In chapter 3 several motivating scenarios were introduces, with the goal to familiarize the reader with the classes of integration problems that the semantic conflict analysis should be able to tackle. Chapter 4 briefly revisited the overall BIZYCLE philosophy for conflict analysis, and sketched structure, behavior, property and communication analysis. The role of the semantic conflict analysis was also positioned within the overall framework. Chapter 5 introduced the application of the semantic annotations, defining the level and element which can be annotated as well as annotation types. Finally, in chapter 6 the actual semantic conflict analysis algorithm was described in detail, including the exemplary test implementation in Prolog.

References

[1] ATL: Atlas Transformation Language User Manual. *http://www.eclipse.org/m2m/atl/doc/ATL_User_Manual[v0.7].pdf*, 2006.

[2] Allegrograph, 2008. http://agraph.franz.com/allegrograph/.

[3] Bprolog, 2008. http://www.probp.com/.

[4] Flora-2: An object-oriented knowledge base language, 2008. http://flora.sourceforge.net/.

[5] Hoolet, 2008. http://owl.man.ac.uk/hoolet/.

[6] Jena semantic web framework, 2008. http://jena.sourceforge.net/.

[7] Kaon2, 2008. http://kaon2.semanticweb.org/.

[8] Owl api, 2008. http://owlapi.sourceforge.net/.

[9] Swi-prolog, 2008. http://www.swi-prolog.org/.

[10] SWRL: A Semantic Web Rule Language Combining OWL and RuleML, 2008. http://www.w3.org/Submission/SWRL/.

[11] Thea: An OWL library for (SWI) Prolog, 2008. http://www.semanticweb.gr/TheaOWLLib/.

[12] Triple Reasoning and Rule Entailment Engine, 2008. http://www.ontotext.com/trree/index.html.

[13] tuprolog, 2008. http://alice.unibo.it/xwiki/bin/view/Tuprolog/.

[14] Visual prolog, 2008. http://www.visual-prolog.com/.

[15] Web ontology language, 2008. http://www.w3.org/2004/OWL/.

[16] Xsb, 2008. http://xsb.sourceforge.net/.

[17] Yaprolog, 2008. http://www.dcc.fc.up.pt/ vsc/Yap/.

[18] Y. Arens, C.-N. Hsu, and C. A. Knoblock. Query processing in the SIMS information mediator. In M. N. Huhns and M. P. Singh, editors, *Readings in Agents*, pages 82–90. Morgan Kaufmann, San Francisco, CA, USA, 1997.

[19] F. Baader and U. Sattler. An overview of tableau algorithms for description logics. *Studia Logica*, 69:2001, 2000.

[20] N. Boudjlida and H. Panetto. Annotation of enterprise models for interoperability purposes. In *Proceedings of the IWAISE 2008*, 2008.

[21] S. Chawathe, H. Garcia-molina, J. Hammer, K. Irel, Y. Papakonstantinou, J. Ullman, and J. Widom. The tsimmis project: Integration of heterogeneous information sources. In *Journal of Intelligent Information Systems*, pages 7–18, 1994.

[22] W. Chen, M. Kifer, and D. S. Warren. Hilog: A foundation for higher-order logic programming. *Journal of Logic Programming*, 15:187–230, 1993.

[23] M. D. D. Fabro, J. Bézivin, F. Jouault, E. Breton, and G. Gueltas. AMW: a generic model weaver. In *Proceedings of IDM05*, 2005.

[24] D. Fensel, S. Decker, M. Erdmann, and R. Studer. Ontobroker: The very high idea. In *FLAIRS '98: 11th International Flairs Conference, Sanibal Island, Florida*, 1998.

[25] C. H. Goh. *Representing and Reasoning about Semantic Conflicts in Heterogeneous Information Systems.* PhD thesis, Massachusetts InstituteTechnology, 1997.

[26] V. Haarslev and R. Mller. Racer: A core inference engine for the semantic web. In *In 2nd International Workshop on Evaluation of Ontology-based Tools (EON-2003), Sanibel Island, FL*, pages 27–36, 2003.

[27] V. Haarslev, R. Mller, and M. Wessel. Querying the semantic web with racer + nrql. 2004.

[28] P. Hoffmann. *Design of a model-based message transformation language.* Diploma thesis, TU Berlin, 2008.

[29] G. Hohpe and B. Woolf. *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions.* Addison-Wesley, 2003.

[30] M. Jang and J.-C. Sohn. Bossam: An extended rule engine for owl inferencing. In *RuleML 2004*, volume 3323 of *LNCS*, pages 128–138. Springer Berlin / Heidelberg, 2004.

[31] F. v. H. Jeen Broekstra, Arjohn Kampman. Sesame: A generic architecture for storing and querying rdf and rdf schema. In *The Semantic Web ISWC 2002*, volume 2342 of *LNCS*, pages 54–68. Springer Berlin / Heidelberg, 2002.

[32] M. Kifer, G. Lausen, and J. Wu. Logical foundations of object-oriented and frame-based languages. *Journal of the ACM*, 42:741–843, 1995.

[33] A. Kiryakov, D. Ognyanov, and D. Manov. Owlim a pragmatic semantic repository for owl. In *WISE 2005 Workshops*, volume 3807 of *LNCS*, pages 182–192. Springer Berlin / Heidelberg, 2005.

[34] A. Kiryakov, B. Popov, D. Ognyanoff, D. Manov, A. Kirilov, and M. Goranov. Semantic annotation, indexing, and retrieval. In *International Semantic Web Conference*, 2003.

[35] D. Kumpe, G. Bauhoff, H. Agt, N. Milanovic, and R. Kutsche. Bizycle metamodelds: Foundation for model-based software and data integration. Technical report, TU Berlin, 2008.

[36] R. Kutsche and N. Milanovic. (Meta-)Models, Tools and Infrastructures for Business Application Integration. In *UNISCON 2008*. Springer Verlag, 2008.

[37] R. Kutsche, N. Milanovic, G. Bauhoff, T. Baum, M. Cartsburg, D. Kumpe, and J. Widiker. BIZYCLE: Model-based Interoperability Platform for Software and Data Integration. In *Proceedings of the MDTPI at ECMDA*, 2008.

[38] A. Leicher. *Analysis of Compositional Conflicts in Component-Based Systems.* PhD thesis, TU Berlin, Computergesttzte InformationsSysteme (CIS), 2005.

[39] C. Li and T. W. Ling. Owl-based semantic conflicts detection and resolution for data interoperability. In *Conceptual Modeling for Advanced Application Domains*, volume 3289 of *LNCS*, pages 266–277. Springer Berlin / Heidelberg, 2004.

[40] C. Mateos, M. Crasso, A. Zunino, and M. Campo. Supporting ontology-based semantic matching of web services in movilog. In *Advances in Artificial Intelligence - IBERAMIA-SBIA 2006*, volume 4140 of *LNCS*, pages 390–399. Springer Berlin / Heidelberg, 2006.

[41] N. Milanovic, R. Kutsche, T. Baum, M. Cartsburg, H. Elmasgunes, M. Pohl, and J. Widiker. Model & Metamodel, Metadata and Document Repository for Software and Data Integration. In *Proceedings of the ACM/IEEE MODELS*, 2008.

[42] C. F. Naiman and A. M. Ouksel. A classification of semantic conflicts in heterogeneous database systems. In *WITS '92: Selected papers of the workshop on Information technologies and systems*, pages 167–193, Norwood, NJ, USA, 1995. Ablex Publishing Corp.

[43] A. Patil, S. Oundhakar, A. Sheth, and K. Verma. Meteor-s web service annotation framework. In *WWW '04: Proceedings of the 13th international conference on World Wide Web*, pages 553–562, New York, NY, USA, 2004. ACM.

[44] E. Pulier and H. Taylor. *Understanding Enterprise SOA.* Manning, 2006.

[45] E. Sirin, B. Parsia, B. C. Grau, A. Kalyanpur, and Y. Katz. Pellet: A practical owl-dl reasoner. *Web Semant.*, 5(2):51–53, 2007.

[46] D. Tsarkov and I. Horrocks. Fact++ description logic reasoner: System description. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 4130 LNAI:292–297, 2006.

[47] D. Tsarkov, A. Riazanov, S. Bechhofer, and I. Horrocks. Using vampire to reason with owl. In *The Semantic Web ISWC 2004*, volume 3298 of *LNCS*, pages 471–485. Springer Berlin / Heidelberg, 2004.

[48] M. Uschold and M. Grninger. Ontologies: Principles, methods and applications. In *Knowledge Engineering Review*, pages 93–155, 1996.

[49] H. Wache, T. Vgele, U. Visser, H. Stuckenschmidt, G. Schuster, H. Neumann, and S. Hbner. Ontology-based integration of information - a survey of existing approaches. pages 108–117, 2001.

[50] Y. Zou, T. Finin, and H. Chen. F-owl: an inference engine for the semantic web. In *The Semantic Web ISWC 2002*, volume 3228 of *LNCS*, pages 238–248. Springer Berlin / Heidelberg, 2004.