TECHNISCHEN UNIVERSITÄT MÜNCHEN

Forschungs- und Lehreinheit I

Angewandte Softwaretechnik

Ontology-based Model Integration for the Conceptual Design of Aircraft

Martin Glas

Vollständiger Abdruck der von der Fakultät für Informatik der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Vorsitzender:	Univ.-Prof. Dr. Ernst W. Mayr
Prüfer der Dissertation:	1. Univ.-Prof. Bernd Brügge, Ph.D.
	2. Univ.-Prof. Dr. Mirko Hornung

Die Dissertation wurde am 31.01.2013 bei der Technischen Universität München eingereicht und durch die Fakultät für Informatik am 17.04.2013 angenommen.

Acknowledgements

I am deeply grateful to Prof. Bernd Brügge, Ph.D. for his advice and encouragement during the development of this dissertation.
I am also very thankful to Prof. Mirko Hornung and Dr. Gernot Stenz for their trust in and support of my dissertation at Bauhaus Luftfahrt.
I would also like to thank Dr. Steffen Prochnow, Dr. Daniel Ratiu, Dr. Sven Ziemer, and the colleagues at the Chair of Applied Software Engineering for their valuable and constructive feedback.
Last but not least, I would like to thank my co-workers at Bauhaus Luftfahrt for their belief in interdisciplinary research, in particular Clément Pornet, Dr. Arne Seitz, Hans-Jörg Steiner, and Dr. Kerstin Wieczorek, who readily contributed their time and expertise as participants and user interface reviewers for my case studies.

Abstract

The development of a new aircraft begins with the conceptual design phase which aims to incorporate the latest technologies and methods into a novel design and its assessment. The development process is usually distributed across several teams which concurrently create models with overlapping content. If overlapping model parts are refined in one model without immediate propagation of the refinements to the other models, they become inconsistent. Exchanging model parts is, however, hampered due to the heterogeneity of these models resulting from discipline-specific modeling approaches. Currently, there are two approaches to tackling this problem: a clear separation of concerns between models to avoid overlapping content and standardized data formats which allow designers to exchange model parts. However, clear separation of design models is difficult to establish during conceptual aircraft design, as the contributing disciplines have considerably overlapping concerns in this phase. Accordingly, overlapping content of design models is unavoidable. Therefore, the problem of consistency and model exchange is addressed by standardized data formats which are interpreted by different tools in the same way. However, conceptual aircraft designers mostly use generic data formats which enable the high degree of flexibility required for this early and explorative phase of design. However, as domain-specific concepts and structures are deliberately not specified by these formats, the meaning of object names and the model decomposition remains implicit knowledge. Therefore, overlapping content and conflicts between models cannot be identified automatically. As a consequence, the maintenance of consistency remains an inefficient and error-prone task. Furthermore, the exchange of concept model parts between disciplines contributing to the same project, and the reuse of existing legacy models is still not well supported by tools.

This dissertation describes a semiautomated ontology-based approach for model integration. An essential part of this approach is the Oida framework which uses a domain-specific reference ontology to facilitate semiautomated consolidation of overlapping content and exchange of selected non-overlapping content between discipline-specific models. Thereby, the owners of the different models map their model elements to the reference ontology. The Oida framework evaluates these mappings, identifies matching model elements, and supports the model owners in resolving conflicts and in optionally importing parts from other models. These capabilities of the Oida framework were evaluated in five quasi-experiments and six case studies using a prototypical implementation of the Oida framework and real sample models originating from different conceptual aircraft design tools. The aircraft designers who operated the prototype during case studies were able to use the framework without help and considered the automatically generated results to be correct from their professional point of view. The solution presented in this dissertation not only contributes to a more automated integration of aircraft models but also to a better collaboration during the interdisciplinary process of conceptual aircraft design.

Zusammenfassung

Die Entwicklung eines neuen Flugzeugskonzepts ist ein interdisziplinärer Prozess, in dem so früh wie möglich neueste Technologien und Methoden aus den verschiedenen Fachbereichen in den Entwurf einfließen. Da die Modelle, welche in den verschiedenen Fachbereichen entstehen, gewöhnlich eine große inhaltliche Überlappung aufweisen und gleichzeitig getrennt voneinander mit unterschiedlichem Schwerpunkt verfeinert werden, entstehen Inkonsistenzen. Außerdem erschwert die Heterogenität der Modelle, die auf Grund von unterschiedlichen Herangehensweisen der Fachbereiche entsteht, einen effizienten Modellaustausch. Gegenwärtig gibt es zwei Ansätze, diesen Problemen zu begegnen: Eine starke inhaltliche Abgrenzung von Modelle und standardisierte Datenformate, um in den unterschiedlichen Fachbereichen dieselben Daten verwenden zu können. Eine stärkere inhaltliche Abgrenzung von Modellen, ist insbesondere im Flugzeugkonzeptentwurf schwer möglich, da sich hier die beitragenden Fachbereiche stark überschneiden. Inhaltliche Überlappungen zwischen den Modellen sind daher nicht vermeidbar. Daher versucht man durch den Einsatz von standardisierten Datenformaten Konsistenz und Datenaustausch zwischen den Modellen effizient zu realisieren, da sie von Werkzeugen gleich interpretiert werden. Allerdings werden im Fugzeugkonzeptentwurf eher generische Standarddatenformate verwendet, um ein hohes Maß an Flexibilität zu ermöglichen. Da diese Formate bewusst keine konkreten Konzepte und Strukturen aus den Fachbereichen definieren, bleibt die Semantik der Namensgebung und des Modellaufbaus implizites Wissen. Inhaltliche Überlappungen und Konflikte können so nicht automatisch erkannt werden. Die daher erforderliche manuelle Arbeit macht den Abgleich von Konzeptmodellen ineffizient und fehlerträchtig. Außerdem wird der gegenseitige Modellaustauch zwischen den Fachbereichen desselben Projekts oder die Wiederverwendung von existierenden Modellen aus vorhergehenden Projekten von Werkzeugen kaum unterstützt.

In dieser Dissertation wird ein Ontologie-basierter Ansatz zur Modellintegration vorgestellt. Ein essenzieller Bestandteil dieses Ansatzes ist das Oida Framework, welches eine fachgebietsspezifische Referenzontologie einsetzt, um den Automatisierungsgrad bei der Integration von Systemstrukturmodellen aus dem Flugzeugentwurf zu erhöhen. Jeder Modellverwalter verknüpft sein Modell mit Konzepten der Referenzontologie. Das Oida Framework wertet diese Verknüpfungen aus und erkennt so automatisch inhaltliche Überlappungen und Konflikte. Darüber hinaus ermöglicht das Framework dem Modellverwalter, augewählte Teile eines anderen Modells in sein Modell zu übertragen. Diese Fähigkeiten des Oida Frameworks wurden durch eine prototypische Implementierung in sechs Fallstudien und fünf Quasi-Experimenten mit Ergebnissen evaluiert. Dabei wurden echte Flugzeugmodelle als Testobjekte verwendet, die aus Werkzeugen mit unterschiedlichendem inhaltlichem Schwerpunkt stammen. Die Luftfahrtingenieure, welche an diesen Fallstudien als Modellverwalter teilnahmen, konnten den Prozess weitgehend selbständig durchführen und hielten die automatisch generierten Ergebnisse aus fachlicher Sicht für korrekt. Die vorgestellte Lösung steigert den Automatisierungsgrad bei der Modellintegration von Flugzeugkonzeptmodellen wodurch die Zusammenarbeit zwischen den Fachdisziplinen im Flugzeugentwurfsprozess wesentlich verbessert werden kann.

Contents

Typographical Conventions

Throughout this dissertation the following conventions are used:

- Citations are given in a comprehensive form (e.g. [ABC08]), indicating the first three authors (e.g. Alpha, Bravo, Charlie) of an article by capital letters followed by the year of publication (2008). If a "+" appears in the citation, then more than three authors have contributed. In the case of a single author, the first letter of his last name is written in capitals, followed by two further lower case letters.
- The Unified Modeling Language (UML) is used for diagrams illustrating software components.
- The typewriter typeface is used for names of software analysis and design objects, such as `RenamerStrategy`.
- Other technical terms and concept terms are written in italics when they appear for the first time.
- Lower capitals are used for product names, such as Eclipse.

1. Introduction

The development process which begins with a first product idea and eventually leads to the blueprints for a new aircraft is usually divided into three phases: conceptual, preliminary, and detailed design. According to Raymer [Ray06] the objective of the conceptual design phase is the development of layouts and assessments of distinct design alternatives addressing a common set of requirements. Thereby, the design space is explored by an iterative process of requirements engineering, analysis, sizing, optimization, and downselection of technologies and architectures. As depicted in Figure 1.1, the aircraft model is refined involving an increasing number of disciplines specialized in certain aspects of aircraft design. The next phase begins after one concept has been selected to be elaborated and scrutinized in the preliminary design phase. In order to prepare for the decision on whether to build the product, the selected design alternative is refined and validated by more sophisticated numerical methods and small-scale physical experiments. If the decision is made to build the aircraft, the following detailed design phase results in all the documents required for the realization of the new aircraft.

Compared to the later design phases, conceptual aircraft design has a high level of design freedom but a low level of model detail and fidelity as it has to rely entirely on simulation and legacy models which have been validated and calibrated against previously built products. However, these legacy models need to be adapted to the new product by reconfiguration and extension. Modification and recombination resulting from new features of the aircraft require a revalidation of the newly created concept model to ensure its credibility. Due to the intentional volatility of the layout during conceptual design, conceptual models are not validated by physical experiments, such as wind tunnel tests. Instead, the credibility of the concept model is increased by applying more sophisticated estimation and simulation methods. The efficient application of these methods requires expert knowledge in certain disciplines. In general, a *discipline* is a part of a scientific domain. The context of conceptual aircraft design is typically associated with engineering disciplines, such as aerodynamics, structural mechanics, and thermodynamics. However, an aircraft concept is also driven by non-engineering disciplines, such as economics and psychology. Each of these disciplines models an aircraft with overlapping scope, thus sharing methods, terminology, and tools. For instance, most disciplines model the outer shape of the aircraft. Aerodynamics examines effects of fluids on the outer shape of the aircraft while the inner structure is generally not relevant. Structural mechanics focuses on the effects of loads to the inner structure of the aircraft which implicitly defines the outer shape. Accordingly, the discipline of structural mechanics shares model parts with aerodynamics such as aerodynamic loads, but for a different purpose. Currently, most aircraft propulsion systems are based on the conversion of thermal to mechanical energy. Therefore, the discipline of thermodynamics is the main driver of the propulsion sys-

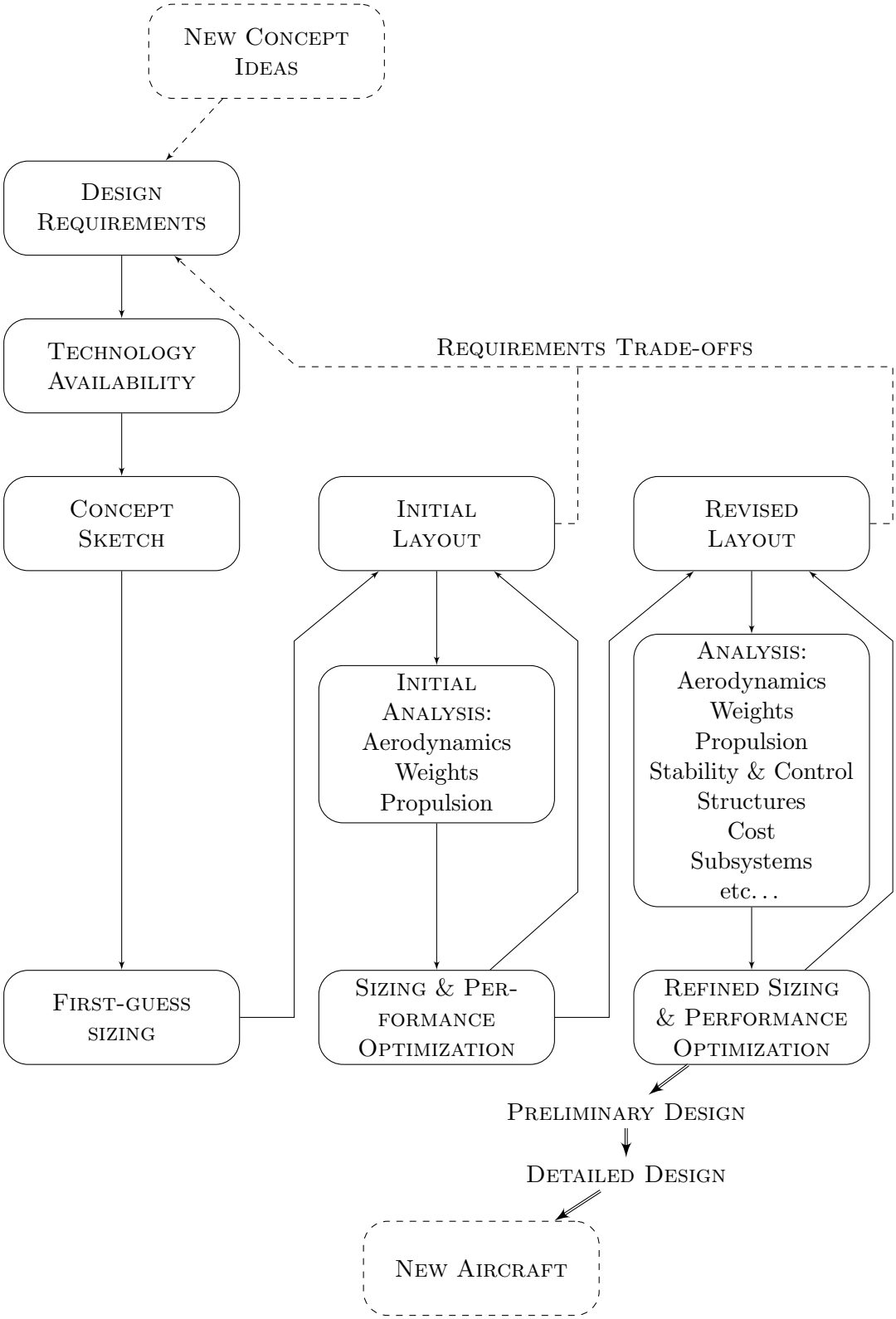

Figure 1.1.: Aircraft conceptual design process according to Raymer [Ray06]

tem. Aircraft components outside the propulsion system are only relevant as far as they influence the flow field of matter and energy of the propulsion system, e.g., the outer shape near the engine mountings or the inner structure of mechanical joints. Even these relatively small examples demonstrate that certain properties of the aircraft need to be shared among models during aircraft design.

If distributed overlapping models are refined without immediate propagation of modifications to all models, they become inconsistent towards each other. The risk of inconsistency is aggravated if the employed design methods deliver results after different time periods. For instance, statistics-based performance estimation methods take a considerably shorter time to yield a result than physics-based numerical methods. If a design team works using data which are inconsistent with the data of another team, their model refinements and design assessments may become obsolete if the conflict is not resolved in favor of their variant. Concurrent model refinement also leads to similar but incompatible variants which are created in parallel. During the early phases of development, parallel refinements and incompatible variants are often pursued to explore the design space collaboratively. However, as the process progresses towards a more mature state of the product design, inconsistencies stemming from this explorative approach become increasingly undesirable for two reasons. First, if the results of other teams cannot be reused, the benefit of cross-fertilization between concurrently working interdisciplinary design teams is not realized. Second, if equivalent variants are not explicitly coupled across the different design models, changes and especially exclusions of variants are not efficiently propagated to all models. The more time elapses between the integration of the distributed models, the more expensive the integration becomes.

A more frequent integration of all design models has two advantages: First, the differences between models are less complex and the amount of work which is potentially obsolete due to inconsistencies is smaller. Second, costly manual mappings between equivalent data sets from different models are reduced as tool independent data exchange based on a common data schema requires considerably fewer mappings.

The following Section 1.1 gives an overview of current practices addressing these problems. Subsequently, Section 1.2 introduces the approach pursued in this dissertation by three characteristic use cases and describes the research method.

1.1. Current Practice

There have been several initiatives supported by industry and academia to develop and establish tool independent standard formats. In the following, STEP, SysML, and CPACS are examined briefly in particular regarding their extensibility by the user which is an essential capability in conceptual aircraft design. *De facto* standard formats which are based on commercial tools, such as Matlab or Microsoft Excel, are not considered, as they have not been developed in an open standardization process.

The STandard for the Exchange of Product Data (STEP)[ISO94] defines a data model and specifies entities from different engineering domains including 3D geometry and management of organizations. According to Peak et al. [Pea+04], STEP is difficult

to extend for an individual solution outside the standard approval process. A part of the STEP standard is the data modeling language EXPRESS. There are different representation standards for the EXPRESS language. ISO 10303-21 defines an ASCI character-based syntax whereas ISO 10303-28 specifies an XML schema. ISO 10303-25 specifies bindings between Express and XMI which is used as an XML syntax for UML models. Aircraft concepts are usually not modeled using STEP as it is not an open standard which can be extended by the user without losing tool support. Indeed, STEP is commonly used to transfer geometry data, as most CAD tools support the standard. However, the mapping of the entire tool-specific geometry model to the STEP standard is usually not bijective. Therefore, round-trip engineering via STEP can lead to data loss.

The Unified Modeling Language (UML) is predominantly used in the domain of software engineering and well supported by several Computer Aided Software Engineering (CASE) tools. The user can extend UML by profiles. For instance, the Systems Modeling Language (SysML) [Gro12] was developed as a profile for systems engineering. Unlike STEP, SysML and UML are not designed to mediate between different tools as an exchange data format. They are standardized modeling languages which system developers can use directly throughout the development process. Currently, SysML is commonly used in academia and industry especially for the development of embedded systems. For instance, spacecraft designers use SysML not only for the development of embedded systems but also for the overall spacecraft. In contrast, designers of aircraft concepts have rarely adopted SysML because its concepts are not well applicable to modeling essential features of an aircraft and its mission.

The German National Aerospace Center (DLR) has recently issued the Common Parametric Aircraft Configuration Schema (CPACS), an XML schema for data exchange between aerospace related tools [Böh12]. The schema defines standard data types and data structures which are commonly used in aerospace models. Furthermore, aircraft designers are provided with aircraft-specific concepts. Thereby, the designers can model an aircraft complying to a schema which has standardized semantics by convention. The data structure, however, is limited to the conventional civil transport aircraft configuration. If an aircraft designer wants to model unconventional aircraft components and configurations, such as electric motors or a blended wing body configuration, he cannot extend the schema in such a way that other clients can interpret it unambiguously.

STEP, SysML, and CPACS define schemata of data types and a standard decomposition of exchange files. The semantics of the data are defined implicitly by convention. However, none of these standards allows the user of the schema to define user extensions formally, e.g., by logical statements, which can be interpreted by existing tools. Therefore, the standard can only be extended by an agreement on a new schema which has to be ratified by tool developers. An exchange standard for a particular domain, such as conceptual aircraft, design is created in a trade-off between domain specificity and general applicability. The more concepts a schema comprises the more difficult it is to attain an agreement to a standard. Narrowing the scope is also not viable for an exchange standard in a multidisciplinary development process.

Within the context of conceptual aircraft design the aforementioned exchange formats are supported weakly by tools regarding aircraft specific concepts. Therefore, the designers have to manually map every data object to an equivalent object in the exchange format.

More generic *de facto* standard model data formats provided by commercial tools like MATLAB or EXCEL are easier to agree upon. However, a generic standard allows the designers to interpret the format's constructs differently. In particular, a generic format allows designers to use different naming conventions and data model decomposition strategies. Tools depend on formal semantics of the standard and its extension mechanism to allow designers to adapt a tool to their particular domain. The current modeling languages and exchange standards are not supported by formal semantics. Accordingly, domain-specific extensions of generic modeling languages like SYSML for the domain of aircraft design or the extension of existing domain-specific standards like CPACS for unconventional system architectures are mostly established by conventions and style guides within an organization.

In a design environment where the meanings of data objects are more informally defined than explicitly specified by the exchange format, tools hardly support the keeping of distributed design models consistent, especially in the likely event that the structure of the designed aircraft is changed fundamentally. As a consequence, the matching between objects from different design models is currently a tedious and error-prone task for domain experts. Furthermore, this practice limits the model complexity and flexibility supported by tools, which in turn limits aircraft designers in their exploration of the design space and in the evaluation of novel concepts.

1.2. Ontology-based Model Integration: Overview

Ontology-based model integration supports an interdisciplinary development process by making discipline-specific models consistent and facilitating exchange of model parts between them. The following section defines the basic use cases of ontology-based model integration.

During the conceptual aircraft design process different models are created which have overlapping scopes but are focused on specific aspects of the aircraft concept. In general, each model has a Model Owner. The Model Owners have to ensure that their respective model is not only consistent with the other models within the interdisciplinary design process, but also have to decide whether to import more detailed parts from another model. This task is called model integration in order to reflect that models are not only consolidated but partly exchanged. Model integration is similar to the model merge operation described by Brunet et al. [Bru+06].

Definition 1 *The operation merge* : $model \times model \times relationship \rightarrow relationship$ *creates a consistent union of two given models. Model merge includes the resolution of conflicts between overlapping elements from the source models.*

Model merge requires a model match operation employing a matching criterion.

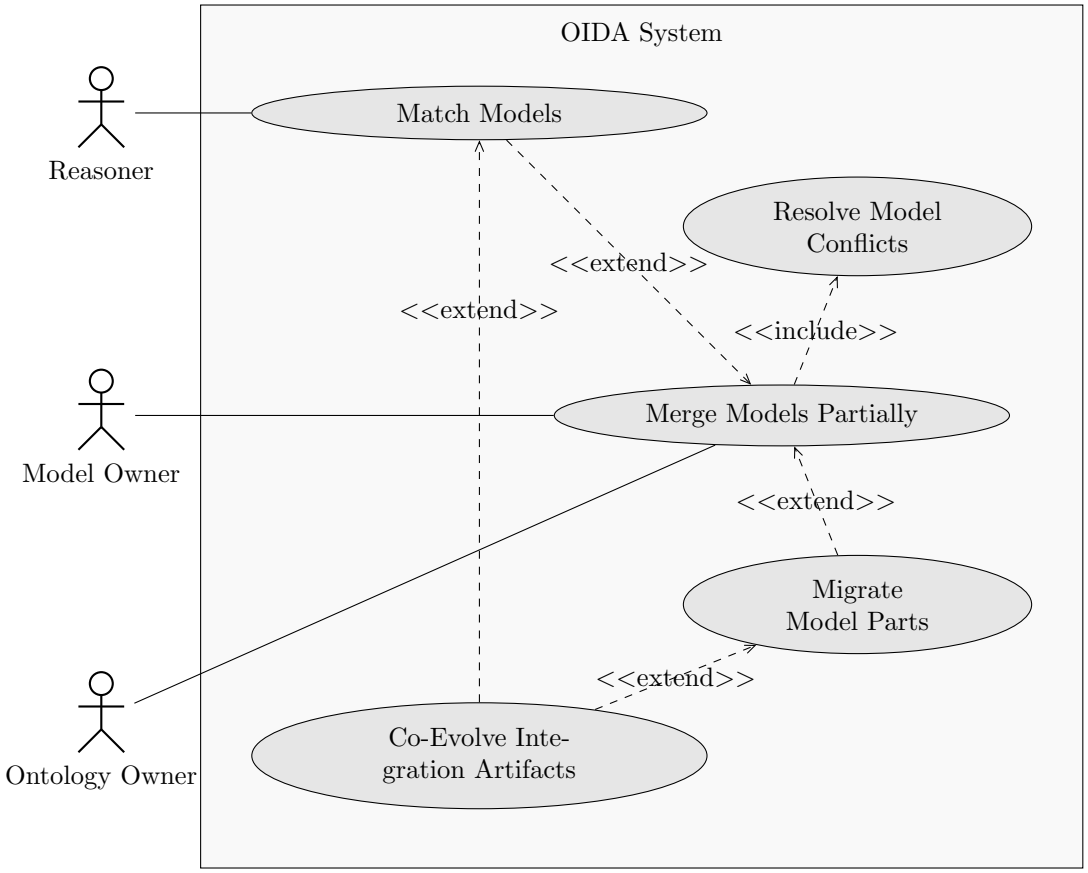

Figure 1.2.: Diagram of the top-level use cases of model integration in conceptual aircraft design

Definition 2 *The operation match : model × model → relationship creates a mapping between the objects of two models according to a matching criterion*

The challenge for the Model Owner during conceptual aircraft design is to keep the complexity of the model at an appropriate level for the specific focus and problem context and to maintain naming decomposition conventions which are adapted to the specific methodology and tool environment of the model. This challenge is addressed by the model integration capability.

Definition 3 *Model integration between discipline-specific models M_i and M_j is the partial merge of models preserving the specific structure and nomenclature of each model. This is achieved by resolving only conflicts between matched objects and by importing unmatched objects only upon the demand of the respective Model Owner.*

The **O**ntology-based **I**ntegration of **D**ata models for **A**eronautics (OIDA[1]) framework described in this dissertation supports Model Owners in performing model integration.

Internally, the OIDA framework generates new artifacts and relations between them as depicted in Figure 1.3 in a semiautuomated process witch involves the Match Models and Merge Models Partially use case. As model integration has to be performed more then once after periods of independent distributed modification of discipline-specific models, the Co-Evolve Integration Artifacts use case addresses the problem of co-evolution of artifacts which are interrelated by the OIDA framework. In the following, these three main use cases are briefly described as well as the research questions and the scope of this dissertation which have led to the development of the OIDA framework.

1.2.1. Match Models Use Case

A fundamental challenge of model integration is to identify abstract objects which represent the same concrete object across different nomenclatures and decomposition strategies. As elaborated in Chapter 2 this challenge is addressed by a matching criterion that is used to identify commonalities and differences between models. However, in order to ensure that objects are interchangeable between different model contexts of conceptual design, a matching criterion has to be supplemented by user decisions. This interactive matching process is addressed by the Match Models use case. In this use case the models are not directly but indirectly matched via a common reference ontology. Corresponding to Gruber [Gru09], an ontology can be defined as follows

Definition 4 *An ontology is a set of concepts of a certain domain, such as entities and relations. An ontology formally defines the meaning and context of these concepts by logical statements.*

Ontologies are essential artifacts in all use cases of ontology-model integration. Therefore, Match Models involves not only the Model Owner but also the Ontology Owner who is responsible for ensuring especially the quality of the reference ontology. Both actors are supported by a Reasoner, an external system which evaluates the logical statements of ontologies in order to detect inconsistencies and infer new statements.

[1] In ancient Greek οἶδα, *oida* means "I know"

1.2.2. Merge Models Partially

The Match Models use case enables performance of the Merge Models Partially which effectuates the actual model integration. In contrast to the *merge* operation, model integration only ensures that objects which represent the same entity are consistent. Unmatched content is only exchanged upon the demand of the respective Model Owner. Accordingly, Merge Models Partially includes the Resolve Model Conflicts use case whereas Migrate Model Parts is optionally performed.

During the Resolve Model Conflicts use case OIDA computes the differences between the model M_i of the Model Owner and another discipline-specific model M_j which have both been matched before in Match Models. M_j is either a legacy model which has been developed for a previous aircraft concept or is a model concurrently developed by another Model Owner. The Model Owner cannot only decide to resolve the conflict by adopting the values of M_j but can also report an implausible matching to the Ontology Owner.

In the optional Migrate Model Parts use case the Model Owner can choose objects in M_j to be migrated to M_i.

Definition 5 *Model migration is the transfer of model parts from a source model to a target model which have been selected by the Model Owner of the target model. If the metamodel of the source and the target model is not the same, model migration employs model transformation.*

Definition 6 *A model transformation translates objects conformal to a source metamodel MM_i to objects in a target model M_j conformal to a target metamodel MM_j. If both source and target metamodels are different in scope and level of abstraction, a transformation between them is not bijective.*

Accordingly, the main challenge of the Merge Models Partially use case is to overcome the heterogeneity of models which stems not only from different metamodels but also fromS different and naming and decomposition conventions. The OIDA framework realizes this bridge by evaluating equivalence statements from the discipline-specific concepts in M_i to the reference ontology.

1.2.3. Co-Evolve Integration Artifacts Use Case

In an aircraft design process the model data is distributed across different artifacts coupled by transformations. In the Co-Evolution the OIDA framework facilitates the efficient propagation of changes in one artifact to other artifacts in order to maintain consistency.

Ideally, a small change in one artifact requires only small changes in the other artifacts coupled to the changed artifact. The OIDA framework is developed to increase the efficiency of change propagation by exploiting knowledge on small modifications which have been performed on the particular artifacts. Co-Evolution is important for the overall efficiency of a process imposed by a framework like OIDA which introduces additional artifacts, such as ontologies.

1.2.4. Research Questions

This dissertation provides a problem analysis, a solution concept, and an evaluation of the implementation which address the following research questions:

RQ 1: What is the nature of the differences observed between different discipline-specific models representing the same physical system? Discipline-specific models describe the same physical system in slightly different ways. These differences stem from different concepts and approaches applied by the developers. For instance, the geometry of the same aircraft is modeled differently in aerodynamics and structural mechanics as these disciplines focus on different features of the aircraft. Furthermore, the different tools influence the practice of modeling, e.g., by assuming a certain aircraft configuration. A deeper understanding of these differences is the basis for the design of an improved methodology for model integration.

RQ 2: How can a software system automate the integration process of concurrently evolving discipline-specific models? During the model integration process the owners of the different models cannot yet fully rely on a software service which automatically performs the model integration. However, a software system can assist the model owner in finding data representing the same physical object and identify conflicts. At the same time, such a software system must be transparent for the respective Model Owner who has to understand and confirm the differences found between models, and has to decide on conflicts.

RQ 3: How can a software system assist co-evolution of dicipline specific models? During the aircraft design process the integration of design models from different disciplines and associated artifacts has to be performed frequently. Model integration requires additional effort from the Model Owner, such as the matching of a model to a common reference. Therefore, how a software system developed for model integration can decrease this effort if model integration is performed frequently has to be shown.

1.2.5. Scoping

This dissertation is focused on the management of discipline-specific models during the conceptual aircraft design process. The problems of detecting topological inconsistencies, integration of system behavior models and system architecture trades are not covered in this dissertation:

1. Aircraft models represent not only the system attributes but also its system structure. During the distributed conceptual design process, structural conflicts can emerge. A model integration system should extract the information about the system structure from different source models and identify matching and conflicting content. In this dissertation, different meanings of model decomposition are analyzed. However, the OIDA does not address the identification of structural system design conflicts.

2. Product models represent not only the structural aspects of a product system but also its behavior. A model of the system behavior is mostly relevant for the assessment of the operational performance of the product. However, behavioral aspects of the system are not taken into account.

3. One goal of conceptual aircraft design is to generate and assess feasible variants of the product's system architecture. However, especially during the design of civil aircraft, the system architecture of the aircraft is fixed from the beginning. Therefore, this dissertation does not address the management of different architecture variants.

This scope includes essential aspects of the current practice of conceptual aircraft design. In particular, the limitation to the structural model aspects of one aircraft architecture is a reasonable context for the development and evaluation of ontology-based model integration.

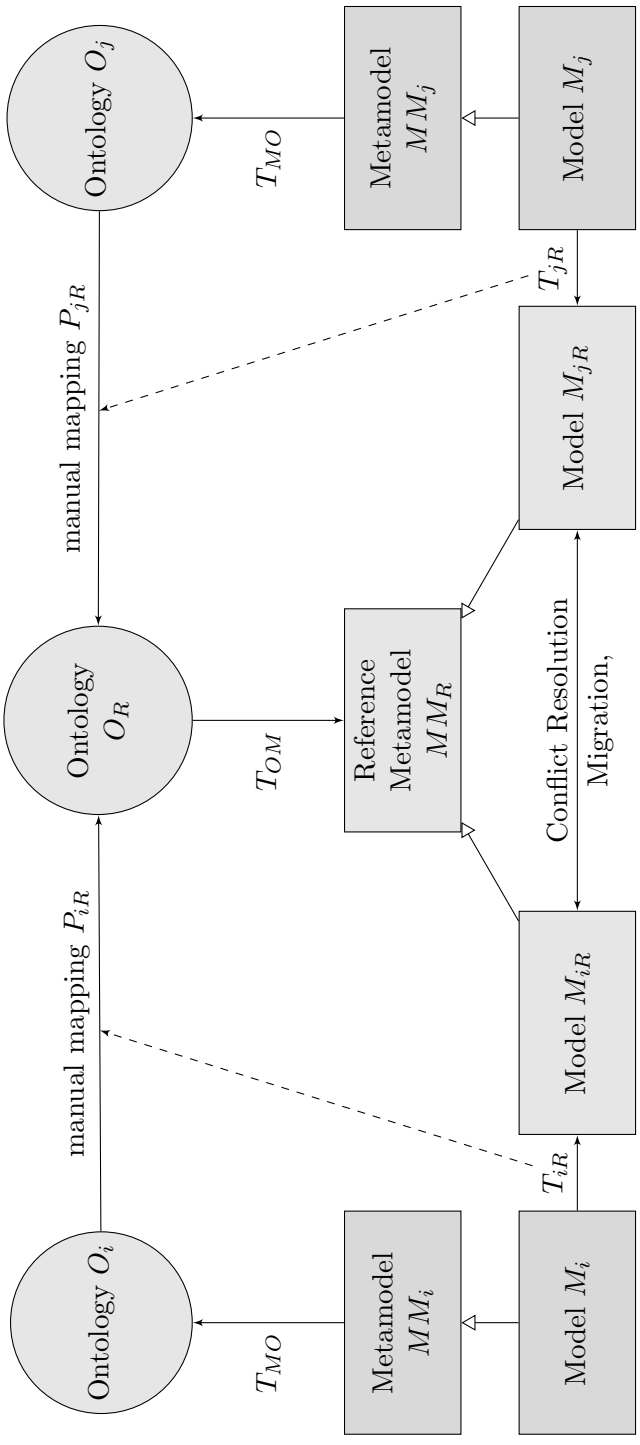

Figure 1.3.: The data flow relations between different artifacts during the ontology-based model integration process. M_i and M_j represent a pair of models from a set of models which have to be integrated.

1.3. Research Process and Outline

The current tool environments for conceptual aircraft design have been developed in a trade-off between flexibility to model new system architectures and components, and the efficient management of consistency and exchange of models from different tools which requires adherence to constraining standards. This dissertation contributes to an improvement of these problems not by a substitution but by an extension of components of existing tools and practices. The basic idea of this dissertation is the mapping of the concrete problem of model integration to an abstract solution of ontology matching. However, the research process evaluates not only the effectivity of the OIDA framework but also efficiency of its application from the perspective of typical users which is a critical criterion for its applicability. Accordingly, this dissertation is structured as follows:

In Chapter 2 the fundamental terminology of the problem analysis and the description of the solution is defined. In particular, concepts of equivalence and mereology are introduced as well as the technological spaces of modeling and ontology.

Chapter 3 translates a concrete problem of concurrent model refinement during the conceptual design of aircraft to the abstract problem of model integration. Thereby, based on an analysis of sample models, the problem is defined by requirements and constraints of the OIDA framework. Additionally, this chapter gives an overview of related work.

Chapter 4 describes the most important capabilities of the OIDA use cases. These use cases are analyzed in Chapter 5 in order to identify participating objects and their relations. The topic of Chapter 6 is the design of the OIDA architecture and its components.

Chapter 7 evaluates whether the abstract solution of ontology-based model integration translates back to the concrete context of conceptual aircraft design.

Chapter 8 summarizes the contributions of this dissertation and discusses topics for future work.

2. Terminology

In the following equivalence, mereology, and technological spaces are defined. The concept of equivalence between objects is introduced stepwise from the concept of identity to equality. Mereology defines kinds of part-whole relationships between entities. Both concepts are especially important for the analysis of the problem addressed in this dissertation. The next section addresses especially the approach and the design of the OIDA by defining and comparing the terminology of modeling and ontology technological spaces.

2.1. Equivalence

Comparing two models regarding matching and distinctive content is an essential capability of model integration as it enables identification of conflicts and content which exists only in one model. This capability requires an operation determining whether two objects from different models represent the same physical object.

As an example consider two objects A and B from the models M_i and M_j. Both models are stored at different locations and can be modified independently. Before it is determined whether A and B represent the same object, it has to be determined whether A and B are actually identical.

Definition 7 *The identity of an object comprises all its properties which distinguishes it from all other objects.*

As an object can only be identical to itself, Dilworth [Dil88] discusses whether identity is actually a kind of relation. Practically, two objects are not identical iff at least one discriminating property is measurable. In the following, we assume that neither M_i and M_j nor A and B are identical as they have at least different names.

For complex objects it is usually impractical to determine identity. However, in a concrete application it is usually sufficient to determine whether two objects are equal assuming an abstract object model.

Definition 8 *Equality is a relation between two objects which can substitute for each other assuming an object model which defines a subset of measurable properties I of an object which are relevant for a certain purpose.*

In the following we assume the object model proposed by Khoshafian and Copeland [KC86]. The properties of an object are attributes and references. An attribute has a type and a value. A reference is a link to another object. The references between a set of objects define a topology of the set which is a graph of objects which disregards

the object attributes. In this object model the behavior of an object is disregarded. According to this object model, two objects can be substituted for each other if they are deep-equal.

Definition 9 *Two objects are deep-equal if their property values are identical, and recursively the property values of all referenced objects.*

If all objects are globally unique, deep-equal objects are also shallow-equal

Definition 10 *Two objects are shallow-equal if their property values are identical.*

Shallow-equality between A and B is less complex to determine than deep-equality as it does not require determination of the identity of property values of all referenced objects recursively. Khoshafian and Copeland [KC86] argue that unique information objects do not necessarily change their identity if their property values or location are modified. Therefore, they propose a more abstract object identity.

Definition 11 *An object identity O is a subset of the object model which is globally unique and independent of property values and location of the object.*

Usually O is implemented as an immutable attribute value that is generated at the creation of every object. If M_i and M_j are only discernible by their location, A and B can have identical O if M_i and M_j are always synchronized, which means that every modification in M_i or M_j is propagated to the other model to ensure that objects with identical O are deep-equal. Now assuming that M_i and M_j are not synchronized, A and B represent the same thing if they are a deep copy of each other.

Definition 12 *Two objects are a deep copy of each other if they are deep-equal except regarding O of the two objects and the recursively referenced objects.*

If A is a deep copy of B, both objects can represent unmodified local copies of an identical ancestor object.

Definition 13 *If A has been created by a deep copy of B, B is the ancestor object of A*

Now it is given that A is not a deep copy of B but equivalent to it.

Definition 14 *Equivalence is a relation between objects which can substitute for each other in a certain context. An equivalence criterion defines a subset E of the object model which can intersect with O. Equivalence between two objects can only be determined if all properties in E can be measured in both objects. Two objects are equivalent if all property values in E are identical in both objects. If no property value in E is identical, the two objects are disjoint.*

The equivalence, equality, and identity are characterized as reflexive, transitive, and symmetric relations. All these relations are reflexive because they always exist from an object to itself. They are all transitive because if one of these relations exists between object A and B, and between B and C, the relation always exists also between A and C. They are all reflexive because if one of these relations exists from A to B it always exists also from B to A. However, these relation characteristics require a bijective mapping of all object properties, i.e. an identical object model. Assuming that M_i and M_j have been created for a different purpose and have a different object model, their objects can be equivalent in a certain context but are not necessarily deep-equal. For instance, 2 and 2.0 are equivalent in a context which disregards decimal digits. However, the example illustrates that a transitive and symmetric equality relation cannot be established for these objects.

2.2. Mereology

Mereology, derived from ancient Greek μέρος meros "part" and λόγος logos "word, rationality", is a discipline which analyzes and describes different kinds of part-whole relationships between objects. Mereology is relevant for model integration as aircraft designers tend to decompose their aircraft model according the the part-whole relationships of the physical aircraft. Thereby, aircraft designers use modeling language constructs which represent containment relations. For instance, UML defines the *composition* association which is a directed relation between two objects. One object can have a composition relation to more than one object but every object can only be contained in one object. If object A is related to B by a composition, B is contained in A. B is destroyed if A is destroyed.

However, containment relations between objects have more differentiated properties. Winston et al. [WCH87] propose a taxonomy of six different kinds of containment which are discriminated by the three properties configurational, homeomeric and invariant. A containment is *configurational* if it relates objects that have a specific functional or structural relationship to one another or to the object they constitute. A containment relation is *homeomeric* if the parts are of the same kind as the whole. A containment relation is *invariant* if the parts can be separated from the whole. Odell [Ode94] adopts these criteria, but proposes a slightly different taxonomy of the following six kinds of composition (see Table 2.1):

- A *component-integral object composition* is a relationship between an entity and its constituting entities. This part-whole relationship can be tangible, abstract or organizational. For example, an engine blade is an integral part of a turbo engine. In contrast to Winston et al., Odell states that if two objects are separated they can no longer have this kind of relation.

- A *material-object composition* interrelates an entity to materials "the entity is made of", e.g., the Boeing B 747 aircraft primary structure is made of aluminium.

	Configurational	Homeomeric	Invariant
Component-integral object	Yes	No	No
Material-object	Yes	No	Yes
Portion-object	Yes	Yes	No
Place-area	Yes	Yes	Yes
Member-bunch	No	No	No
Member-partnership	No	No	Yes

Table 2.1.: Six kinds of composition according to Odell

- A *portion-object composition* describes a composition where the parts are all of the same kind as the whole. For example, a foot is a part of a nautical mile. Both quantities are lengths.

- A *place-area composition* is a relation of a geographic spot to a geographic area that completely surrounds it. For instance, the threshold is part of the runway.

- A *member-bunch composition* is a relation of entities to a group they constitute. The member-bunch relation is *accidental*, i.e., the group is not destroyed if a member leaves the group. In the aviation domain, several aircraft are in a member-bunch composition to a particular carrier fleet.

- A *member-partnership composition* is a relation of entities which together form a partnership entity. Thereby, the constituting members are *essential* to the partnership entity. In aeronautics, the pilot and the co-pilot constitute a cockpit crew. If one of the members leaves the crew, the crew ceases to exist.

The examples from aeronautics not only illustrate their relevancy for conceptual aircraft design but also that common modeling language constructs, such as Composition in UML, can represent these relations but do not differentiate between them.

Beyond these composition relations, Odell defines four other kinds of relations which are in practice often mistakenly expressed as part-whole relations:

- A *topological inclusion* of an object into another surrounding object is not a part-whole relationship as long as there is no close relationship between the two objects. For instance, if a person is in a building, the person is not a part of that building.

- If an entity is an instance of a certain class, entity and class are not in a member-bunch relationship but in a *classification inclusion* relationship. For instance, if `Man` has the instance `Bob`, `Man` is not a bunch but an abstract class. Accordingly, `Bob` is not a member of `Man`.

- An *attribution* is the relation of an entity to its attributes. By this relation, attributes are associated with an entity but are not components of it, e.g., if a box is red, the color red is not a part of the box.

- An *attachment* is a relation between linked objects. Linked objects are not necessarily in a part-whole relationship. For example, a lid on a pot is connected and has a functional relationship to the pot but is not part of it.

- *Ownership* is a relation from one entity to another entity which it possesses. If an entity A owns entity B, it does not mean that B is part of A, e.g., if Alice owns a dog, the dog is not part of her.

The occurrence of these pseudo-containment relations is an important symptom of models which represent the same system but with a different decomposition strategy.

2.3. Technological Spaces

According to Kurtev et al. [KBA02] a *technological space* (TS) is a combination of methodologies and tools, which are learned, applied, and further developed by a community of persons. In contrast to disciplines, the technological space concept emphasizes a degree of self sufficiency which allows its members to solve a problem within its boundaries. Basically, the OIDA framework establishes a mapping between the *Modeling TS* and the *Ontology TS*. By this mapping the problem of model integration is transformed to a problem of ontology integration. The problem is solved in the Ontology TS and propagated back to the Modeling TS. In the following, both technological spaces are briefly described by defining some of their fundamental concepts.

Concepts of the Modeling Technological Space

The *Modeling Technological Space* (Modeling TS) is a working environment realizing the *Model Driven Architecture* (MDA) proposed by the Object Management Group (OMG) [Mel+02]. The artifacts of this technological space are models and metamodels. MDA also stipulates transformations between *Platform Dependent Models* (PDMs) and *Platform Independent Models* (PIMs). The basic idea of MDA is to improve the efficiency of developing and maintaining systems by making changes on platform-independent models (PIM) and to generate platform-dependent models (PDM) by these transformations. An important framework of the MDA is the *Meta Object Facility* (*MOF*) [Gro11] which proposes a framework of four layers of abstraction.

The $M0$ layer contains objects from the *real world*. A set of these objects is a system. For instance, physical systems and a running software system exist at the $M0$ level.

The $M1$ layer contain objects which are abstract representations of the state and behavior of real world objects and their relations on the $M0$ level. A set of these objects is a model of a system. For instance, the software source code is a model of a running program.

Objects on the $M2$ layer are abstract specifications of properties and operations of objects and relations at the $M1$ level. A set of these objects is a metamodel. Objects in a model are instances of objects in a metamodel. Thereby, a metamodel can specify a formal language. For instance, the Unified Modeling Language (UML) [OMG11] is specified on the $M2$ layer.

Objects on the $M3$ layer are an abstract specification of objects and relations at the M_2 layer. A set of these objects is called a meta-metamodel. For instance, MOF is on the $M3$ layer. MOF is not only the metamodel of UML on the $M2$ layer but defines also its own metamodel by the fundamental concepts classifier, instance, and reflection. The latter enables the navigation between classifier and instance objects.

The object model specified by the common classifier of an object can be used as an equivalence criterion as the `instanceOf` relation ensures that these properties are measurable in both objects.

Abstraction has the consequence that an identical real object can be represented by objects with a non-identical set of properties. For example, if two have been created from two different metamodels or the same metamodel has been interpreted differently, two objects representing the same real entity can only be determined by matching.

Definition 15 *A match is a relation between objects that are not necessarily equivalent but represent the same object. The relation is determined by a matching criterion which defines a subset S of properties in an equivalence criterion E. Two objects do not match if at least one property which is measurable in both objects is not identical.*

In contrast to equivalence, the match relation is reflexive but not transitive and symmetric. Two matching objects cannot substitute for each other if they are not equivalent. Given that two objects match each other, e.g., (name: `wingspan`, measure: `3`; unit: `m`) matches (name: `span`; measure: `3`), substituting `span` for `wingspan` would reduce information in the name and unit attribute.

A common matching criterion is a common ancestor object. This equivalence criterion determines equivalence between local copies, which have been modified independently. However, to determine a match between two objects if they were created independently or if the ancestor object is not measurable, an equivalence criterion must define necessary and sufficient conditions specific for a concrete application.

If object A and B match but are not equivalent they are in conflict.

Definition 16 *Conflict is a relation which can only exist between matching objects. A conflict criterion C is the relative complement of S in E. If at least one property in C is not identical, the equivalent objects are in conflict.*

Now given that A and B are in conflict, the conflict can be resolved.

Definition 17 *A conflict between two equivalent objects is resolved by modifying every property in C to an identical value.*

Usually a conflict is resolved by copying all values from one of the objects to the other. After a conflict resolution two matched objects are equivalent in a certain context. However, the two objects are only equal if they have the same classifier which is fully covered by E and C. All possible classifiers of a model are defined in its metamodel. Therefore, equality between two models can only be ensured if they have an equal metamodel.

Concepts of the Ontology Technological Space

The *Ontology Technological Space* (Ontology TS) is a combination of methods and tools for knowledge representation and automated reasoning. In particular, an ontology language is used to represent the knowledge in an ontology.

The term *ontology* is derived from the ancient Greek words εἰμί eimi "I am" and λόγος logos "word, rationality". In philosophy it means the study of "being" entities and their structure. In the domain of computer science "[...] an ontology defines a set of representational primitives with which to model a domain of knowledge or discourse"[Gru09]. Accordingly, an ontology defines entities and relations between these entities which are constrained by logical statements. Thereby, the semantics of the entities can be formally defined. The intended application of an ontology is an important feature. For instance, the primary application of the reference ontology in this dissertation is to provide a representation of knowledge in the domain of conceptual aircraft design. A machine-readable ontology representation can be used for automated reasoning. The basic functionality of an automated *reasoner* is the classification of entities. Thereby, a reasoner not only determines the consistency of statements in an ontology but can also infer new statements.

Ontologies can be connected to each other in a hierarchy. For example, an ontology containing application-specific concepts can import another ontology containing more general concepts and use it as an upper ontology [Sch03].

Definition 18 *An upper ontology comprises abstract and generic concepts which can be shared across domains, such as mereology or physical measures.*

Ontologies can be designed with the help of tools or derived from natural language or existing models. An ontology language can express semantic relation by logical statements which can be evaluated by a reasoner.

Model vs. Ontology

Models and ontologies both deal with the representation of physical and virtual concepts. However, models and ontologies are generally used with different intentions [Hen11]. Models are prescriptive representations which are usually based on the *closed world and unique name assumption*, which makes them more adept for systems design and implementation. In contrast, ontologies are descriptive representations which are usually founded on the *open world assumption*. For example, a model and an ontology represent the facts that every person can only have one mother and that there are two persons `Bob` and `Bob`'s mother `Peggy`. Both model and ontology can be queried as to whether

`Margret` is `Bob`'s mother. According to the closed world assumption, the answer for the model is "No", because the model does not contain an explicit statement that `Margret` and `Peggy` are the same person. In contrast, according to the open world assumption, the result of the query to the ontology is "Unknown", because there is no explicit statement that `Peggy` and `Margret` are not the same person. Now the fact that `Margret` is `Bob`'s mother is added to both model and ontology. Based on the unique name assumption the model would become invalid, as `Bob` cannot have two mothers. As the ontology does not assume unique names for entities, the ontology does not become invalid. Additionally, an automated reasoner can evaluate the ontology and infer that Margret and Peggy are the same person. In this example, the ontology is not only more robust than the model dealing with different perspectives to the same object but has derived new knowledge.

Ontologies represent concepts by entities and properties. Entities are classes and individuals who are instances of classes. Properties are relations from entities to other entities or to data types. In contrast to modeling languages, ontology languages allow declaration of an equivalence relation between concepts by an equivalence statement. For instance, an equivalence statement between two individuals (name: `wingspan`; measure: `3`; unit: `m`) and (name: `span`; measure: `3`) has the effect that both objects are treated as (name: `wingspan, span`; measure: `3`; unit: `m`). Accordingly, the conflict resolution and the declaration of equivalence between two matched objects leads to equal concepts even if the concepts have a different set of properties.

3. Problem Definition and Related Work

The problem context addressed in this dissertation is the conceptual aircraft design phase which is software intensive as it has to rely predominantly on virtual product models. The different levels of abstraction and the flexibility of modeling languages allow different naming and decomposition conventions, which inhibits efficient matching objects between different models. However, despite the abstraction of design models their integration is based on the assumption that design models represent the same physical object. Therefore, the following chapter gives an analysis of differences between sample models from conceptual aircraft design. Based on these observations, the problem is defined by stating the requirements and constraints of the OIDA which is put into the context of related work already performed by other researchers.

3.1. Sample Models

The concept of ontology-based model integration was developed and evaluated using three sample models from different tools called SIMCAD, APA, and APD.

These tools were chosen as they cover a variety of typical data models in aircraft conceptual design with respect to scope, complexity, and focus. Although they are generally not used concurrently in the same project, they describe structural aspects of the same type of aircraft. Therefore, they can be used to simulate the situation when three models of the same aircraft are generated by independent design groups.

SIMCAD is a tool which has been developed at the Airbus Future Projects Office for conceptual aircraft design studies. It is basically a collection of scripts executable on the SCILAB platform. SCILAB is an open source numerical calculation environment similar to MATLAB. SIMCAD employs two types of internal data exchange: (1) the processing blocks passing data via function parameters and (2) the processing blocks manipulating data on a globally accessible data structure like reading from and writing on a black board. The *black board* is a data structure which is used as a global look-up tree for aircraft design algorithms, such as weight and performance estimation. Within the black board the naming convention for variables is widely comprehensible for any aircraft design expert. A special SIMCAD script serializes the *black board* using a specific syntax. The script not only encodes the treelike data structure but specifies units for every scalar value. As the *black board* data structure mostly contained structural aspects of the aircraft was used as the SIMCAD sample model.

The Advanced Propulsion Analysis tool (APA) has been developed at Bauhaus Luftfahrt based on MATLAB. It was originally developed by Seitz [Sei12] to compare the

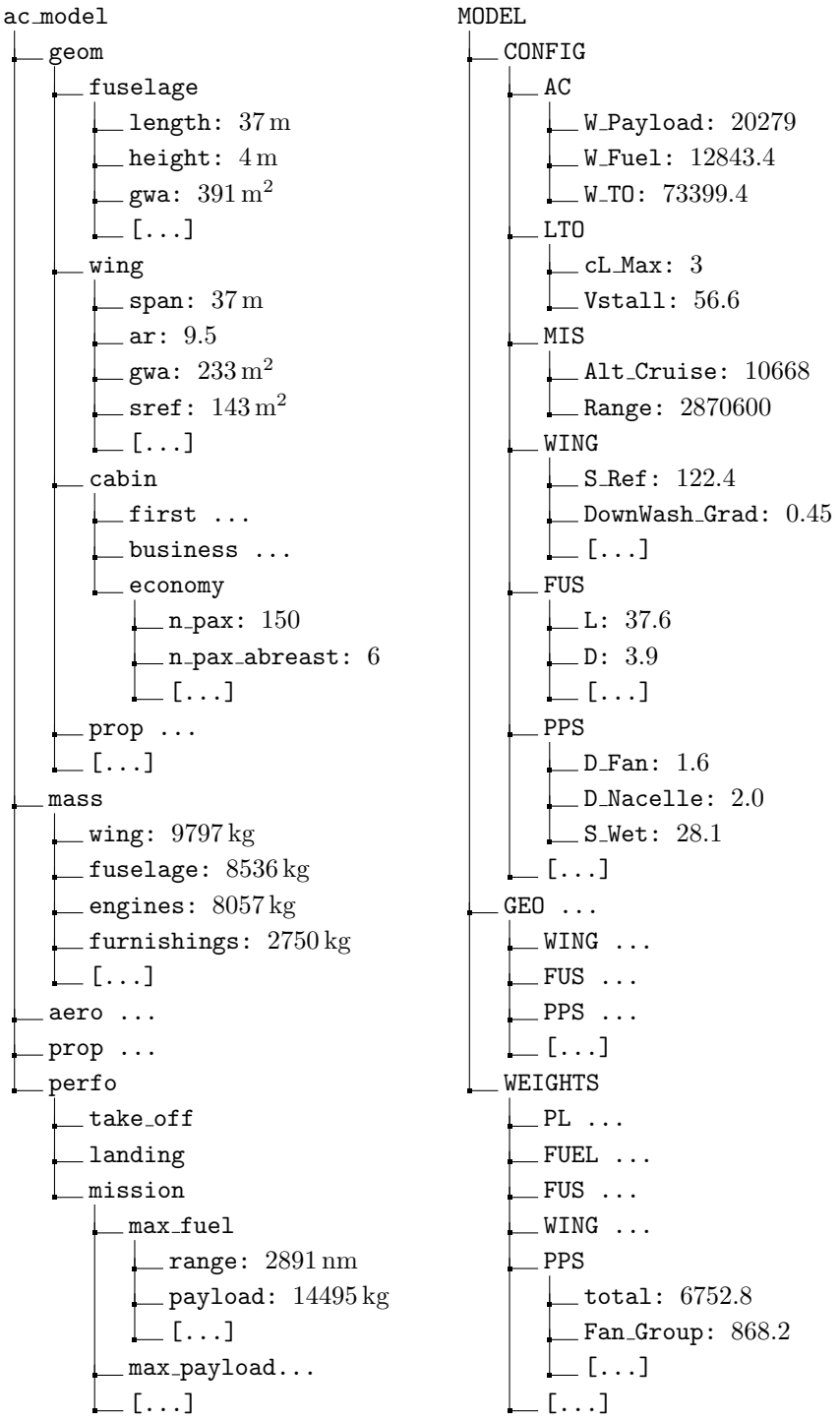

Figure 3.1.: Excerpts of the SIMCAD sample model (left) and the APA sample model (right) which show different interpretations of the same metamodel as well as different naming convention and decomposition strategies while representing a similar aircraft

"Open Rotor" versus the "Ducted Geared Turbo Fan" propulsion concepts on a common basis. Accordingly, the structural aspect of the model covers the overall aircraft but is focused on propulsion-related aspects of the system. APA employs a central data structure representing aircraft system attributes, which are used by the calculation scripts both as data source and target. In accordance with common practice of modeling in MATLAB, the units of measure for each attribute are not specified in the data structure. Instead, a MATLAB script responsible for serialization of this data structure contains a unit specification and short description of each value as comment. The naming convention of variables in APA is mostly oriented to symbols commonly used in aircraft design literature. However, without the comments in a separate script file, the nomenclature is difficult to comprehend. The central data structure contains a considerable amount of data which do not define structural aspects of the aircraft, and thus are beyond the scope of this dissertation, e.g., the results of mission simulations which rather describe the behavior of the system. However, the model objects associated with these aspects are arranged in subtrees of the model which could be clearly separated. The remaining APA central data structure was exported to a file and used as the APA sample model.

The PACE LAB SUITE is a commercial tool developed by PACE LAB for conceptual aircraft design. PACE LAB offers the AIRCRAFT PRELIMINARY DESIGN (APD) plug-in which provides a collection of implemented common preliminary aircraft design methods and models of recent civil transport aircraft, such as the AIRBUS A320. These aircraft models are used to benchmark new aircraft concepts or calibrate new performance estimation methods. Generally, the models cover the overall aircraft and have no focus on a specific subsystem. The aircraft model parameters can be serialized to a comma separated values (CSV) file. Each entry in the CSV file contains the name of the parent object representing the hierarchical model structure. Entries representing scalar values additionally specify the respective unit of measure. The APD parameters also contain data which describe the system behavior. For instance, there are data tables on the specific fuel consumption (SFC) of the engine in different system states. The identification of equivalent state variables like SFC requires not only matching of their dependencies to system states but also interdependencies among system states. However, the integration of system states, which are part of the behavioral model of an aircraft system, is not addressed in this dissertation. Therefore, SFC tables were removed early from the CSV file by filtering representations of scalar values. The result was used as the APD sample model.

The similar structure of the tool data models allows the transformation a tool specific format to a common metamodel. For this conversion, the tool connectors of the Conceptual Design Tool (OPENCDT) [ZGS11] were used which transfer the variable names and the data structure of the original models directly to MOF-compliant JAVA object models. Using this representation, the following comparative observations could be made:

- The sample models exhibit considerable semantic overlap. As a consequence, elements from different source models represent the same physical object. Under

the assumption that the models represent the same aircraft, these overlaps are a potential source of inconsistencies.

- The sample models contain attributes which characterize the modeled aircraft. These characteristics either explicitly describe the *system topology*, e.g., the number of windows, or describe *measurable attributes* of system components. They can be scalar, or be represented in more complex structures, such as vectors, or tensors. Only the SIMCAD and the APD sample model explicitly state units of measure.

- All sample models use the object composition association for different *decomposition strategies*. In the SIMCAD model, for instance, it can be observed, that the containment relation is used for three different kinds of decomposition. There is a geometry tree representing a component-integral object decomposition. The `mass` container represents classification inclusion decomposition. The `take-off` container represents a decomposition oriented to an attribution relation to system states. Only the `geom` container and its child objects have a component-integral object relation which can be classified as containment. The APD model shows the same types of decomposition. The APA consistently uses classification inclusion relations at the first tree level and an attribution relation in the second layer. It can be concluded that the decomposition strategy of all sample models is rather designed more towards efficient access to data entries during the numeric calculation processes within the respective tool than towards data exchange.

- Generally, the name attribute of a model element indicates the physical entity it represents. However, this *naming convention* is not standardized. Especially, in APA the objects are named by abbreviations which are difficult to comprehend without further documentation outside the model.

3.2. Oida Requirements and Constraints

Based on the analysis of the sample models, the OIDA framework has been developed according to the following top-level requirements:

- The entities of all models must be matched with a common reference ontology. Model elements representing data specific to tools or disciplines can be deliberately excluded from the integration process by the model owner. Furthermore, model parts representing non-structural aspects of the aircraft as well as non-scalar attribute values are generally excluded from the mapping process.

- The OIDA framework must facilitate the detection and resolution of conflicts between equivalent model elements from different sample models. Thereby, one of the models is declared to be the Chief Engineering Model which overrules the other models in the case of conflict. This scenario assumes that a chief engineer is entitled to propagate his design decisions to the other models.

- The OIDA framework must give Model Owners an essential role in the ontology-based model integration. In particular, Model Owners must perceive the process of ontology-based model integration as efficient and the result as plausible from the aircraft design expert point of view.
- Repeated performance of the model integration process with small changes in the source models, the reference ontology, or the integration related algorithms must require only small changes on the other coupled artifacts to maintain overall consistency.

The following design constraints have been posed early in the development process in order to foster a simple but meaningful and extensible proof-of-concept:

Reference Ontology It was decided to use an ontology as a common reference for the integration of discipline-specific models. A classification of technical approaches to model integration was described by Noy [Noy04]. She discriminated between ontology-based integration and heuristics-based integration techniques. The latter are commonly used in database schema integration and are based on lexical and structural analysis of the given models. In contrast to these techniques, ontology-based approaches exploit the semantics of relationships between entities. Generally, not only ontology languages but also modeling languages allow defining and constraining entities and links. For instance, UML defines relationships like `Generalization`, `Composition`, and `Attribute`, which can be constrained by OCL. However, the semantics of these relationships are not formally specified. In contrast, an ontology language, such as OWL, can express formal semantics by description logic [BHS08] which can be interpreted by automated reasoners. The resultant reference ontology can potentially be reused in other application contexts.

Simple Equivalence Another early decision was to realize the matching of discipline-specific models exclusively by equivalence statements between ontologies. Weaker nuances of equivalence would allow the domain expert to adequately express relation between similar concepts. However, these nuances of equivalence are not defined in standard ontology languages. Furthermore, application of similarity relations during Match Models requires a shared understanding of similarity among all Model Owners which would add organizational complexity. Therefore, OIDA only uses equivalence relations. This constraint also limits the complexity of the ontology, as the Model Owners only add an equivalence relation between two concepts if they consider them to represent the same object. Furthermore, the limitation on standardized simple equivalence facilitates the development and evaluation of ontology processing capabilities as these relationships are well supported by existing ontology frameworks and reasoners.

Simple Part-whole Relationship As described in Section 2.2 mereology is an important feature of physical systems and the models representing them. Therefore, a simple mereology upper ontology is provided as a common basis for both the reference ontology and ontology-based transformation algorithms. The mereology upper ontology defines

the transitive properties `isPartOf` and its opposite `hasPart` as well as the corresponding subproperties `isPartOf_directly` and `hasPart_directly`. The reference ontology uses these properties as a generic foundation of a component-integral object relation, whereas the ontology-based transformation algorithms asserts the `hasPart_directly` property as a formal representation for the composition association of UML. The detailed design of the mereology upper ontology and its application in the reference ontology is provided in Appendix B.

3.3. Related Work

The application of ontologies for the integration in conceptual aircraft models is based on work already conducted by other researchers who have already successfully applied the consolidation and transformation of models from different sources. However, their approaches do not address the level of heterogeneity and domain specificity pursued in this dissertation.

3.3.1. Tool and Model Integration

Kramler et al. [Kra+06] describe the idea of a semantic infrastructure for tool integration. The metamodels of tools are "lifted" to so called tool ontologies, i.e. the metamodel elements are mapped to ontologies which contain tool-specific concepts. These tool ontologies are "bound" to a generic ontology, i.e. the tool specific concepts are mapped to generic concepts. To give an example from business process modeling, the concept `Action` from UML and `Activity` from BPEL can be bound to the generic concept of `ProcessStep`. The resulting bindings are used to generate model transformations which allow transformation of models from one tool to another. This infrastructure is similar to the OIDA framework. However, analysis of the sample models revealed that the gap between the abstraction levels of the metamodels and the instance models allowed too much ambiguity of the object model regarding its meaning and relation to other model elements. Thus, mapping on the metamodel level proposed by Kramler et al. [Kra+06] is not sufficient for the reliable determination of overlaps between the sample models.

Maalej [Maa10] uses ontologies as an essential component of his solution for tool integration. While his work focuses on seamless work flow integration, his description of specific features of ontologies as enabling technology also applies to the integration of design models. Therefore, his considerations also motivated the design decision to develop an ontology-based approach for model integration.

Del Fabro and Valduriez [DFV07] claim that the efficient generation of model transformation is a key technology for efficient model integration. The authors describe the automated generation of a weaving model which contains a structured mapping between two metamodels. For instance, given two metamodels from business process modelling the waving model would contain a mapping object between the classes `ActionItem` and `Task`. This mapping object would further contain mapping objects between the respective attributes and associations of these classes. The automated generator of such a weaving model identifies synonym relationships between metamodel entities by queries

to the WordNet ontology [Fel10]. However, their solution is only based on metamodel matching, and does not evaluate meaning or context of instance model elements. Furthermore, the solution does not address the situation when WordNet does not contain a concept, a situation which is likely in conceptual aircraft design. In contrast, Oida allows addition of new concepts to the reference ontology during the Match Models use case.

Simon Zayas et al. [SZMA11] apply model matching via a reference representation of the product system model. This reference representation is not realized by an ontology but by a model which is separately created in one of the tools. This reference model allows matching of models from different tools representing different developmental aspects to the same avionics system. However, the reference model strongly depends on a specific tool and development domain which limits its reuse in another application context. Furthermore, the language of the reference model does not have the expressivity of an ontology language regarding semantics and cannot be directly processed by a reasoner. For this reason, as discussed in Section 3.2, the Oida uses an OWL ontology as common reference for integration.

Roser [Ros08] realizes ontology-based model transformation by employing different nuances of equivalence, such as similarity and overlap, for the matching of metamodel elements, as well as OWL and an automated reasoner in order to facilitate the matching of the metamodel concepts. The transformation is generated using the indirect matchings of target and source metamodels of the transformation via a reference ontology. Oida also uses a reference ontology to realize model transformation and matching. The generic data format of sample models allowed a simple transformation of the sample models to the same metamodel using the tool connector capability of OpenCDT. Therefore, the Oida framework addresses the problem of matching different interpretations of the same generic metamodel, which manifests in different naming and decomposition conventions of the sample models. Furthermore, in contrast to Roser's solution which is fully automated, the Oida engages the Model Owners in an interactive process. Thereby, Model Owners not only control the process, but also report errors of automated matching algorithms and extend the reference ontology in the process

Different types of model integration approaches are categorized regarding organizational effort by Jardim-Goncalves et al. [JAS10] by estimating the time required for communication between organizations. They discriminate between slack, unregulated, standard-based, semantic interoperability, and sustainable interoperability. Accordingly, the Oida framework is a *semantic interoperability* type approach which is more efficient than *standard-based interoperability* approach, e.g., CPACS or SysML, if the mapping of the model data to a common ontology is faster than solving data clarification issues for each model exchange by experts. Therefore, in order to support the assessment ontology-based model integration, the evaluation in Chapter 7 investigates the efficiency perceived by Model Owners during the Match Models case studies and the measurable efficiency in Co-Evolve Integration Artifacts quasi-experiments.

3.3.2. Ontology Generation and Matching

Gašević et al. [Gaš+04] aim at a seamless integration of ontology tools with UML-based modeling tools. They present a transformation from the Ontology Definition Metamodel Specification (ODM) [Obj09] to OWL, based on the eXtensible Stylesheet Language Transformation (XSLT). The mapping between UML and OWL which is defined in the annex of the ODM specification served as a template for the model-to-ontology and ontology-to-model transformations of OIDA. However, OIDA is not designed to provide an ontology-based aircraft modeling tool but to facilitate the integration of existing aircraft models employing ontology-based technologies. Therefore, OIDA uses separate specialized frameworks for handling models and ontologies, respectively.

Halpin and Hayes [HH10] describe the history of the concept identity and the consequences of a `sameAs` relation between concepts from different ontologies. The authors argue that this kind of equivalence relation is a strong statement and should be used carefully in distributed information systems, such as the semantic web. Their recommendation opposes the constraint stated in Section 3.2 that the OIDA framework only uses simple equivalence relations. However, the sample models are significantly less complex than the semantic web. Furthermore, the Match Models use case stipulates the Ontology Owner who can advise the Model Owner to declare an equivalence relation in undoubted cases.

Doan et al. [Doa+03] propose a schema-matching algorithm GLUE which uses a set of "learning strategies" for finding the best match between schema items. "Learning" in this context means that an algorithm uses a given structure to generate good propositions for schema matching. The learning strategies used by Doan et al. are comparable with the concept of `EquivalenceFinders` in OIDA described in Chapter 6. For instance, the "content finder" corresponds to the `StructuralEquivalenceFinder`, the "name learner" corresponds to the `NameEquivalenceFinder` but also takes the containers of the respective entity into account. The capability of a "meta-learner", which is basically an aggregation of the aforementioned strategies by a weighted sum, is realized in the OIDA by a BLACKBOARD pattern. GLUE supports complex mappings between ontology entities. For instance, the entity `name` in one ontology can be related to two entities `firstName` and `lastName`. In contrast to GLUE, OIDA's ontology-matching process does not require complex mappings during Match Models as the tool connectors of OPENCDT transform the sample models to a common metamodel.

3.3.3. Evolution of Coupled Artifacts

Herrmannsdoerfer et al. [HBJ09] address the problem of co-evolution of metamodels and their model instances by the EMF EDAPT framework. EMF EDAPT is based on recording the user's modifications to a metamodel which are put into the context of a specified refactoring which the user intends to apply. After certain types of refactorings to a metamodel have been recorded, the framework generates a transformation script which migrates existing instance models of the old version of the metamodel to the new version of the metamodel [HVW11]. The idea of the EMF EDAPT framework to

evaluate known modification to artifacts in order to facilitate the co-evolution of coupled artifacts has inspired the design of the OIDA evolution capabilities. In particular, the OIDA framework provides a prototypical support for a rename refactoring in the reference ontology.

Kögel [Kög11] shows that a system performs significantly better than a text-based model version control process. EMFSTORE, an implementation of operation-based version control, uses recorded change operations on local copies of the model under version control. As change operations reference the metamodel, the conflict detection and conflict resolution recommendations to the user are more specific to the domain context of the user as compared to a text-based version control system, which are agnostic to the metamodel. The software algorithm for conflict detection in OIDA is not operation-based as changes to the discipline-specific models are performed outside of the system boundaries of the OIDA framework.

4. The Oida Functional Model

> "No two manufacturers ever split it up quite the same way and every mechanic is familiar with the problem of the part you can't buy because you can't find it because the manufacturer considers it a part of something else."
>
> Pirsig [Pir74]

In this chapter the main use cases Match Models, Merge Models Partially, and Co-Evolve Integration Artifacts are analyzed in detail in order to identify actors, required capabilities of the Oida framework, and their interaction. In this dissertation the term capability is used according to systems engineering as a high level requirement defining an ability of a system to perform or facilitate a certain task without prescribing a certain solution [HP08]. In particular, Section 4.1 analyses different situations of model matching leading to the ontology-based approach realized by the Oida framework. After a brief description of the Merge Models Partially in Section 4.2, Section 4.3 defines four scenarios which require Co-Evolve Integration Artifacts.

4.1. The Model Matching Capability

The goal of model matching is the identification of elements in different models which represent the same object. This capability is required to determine conflicting objects

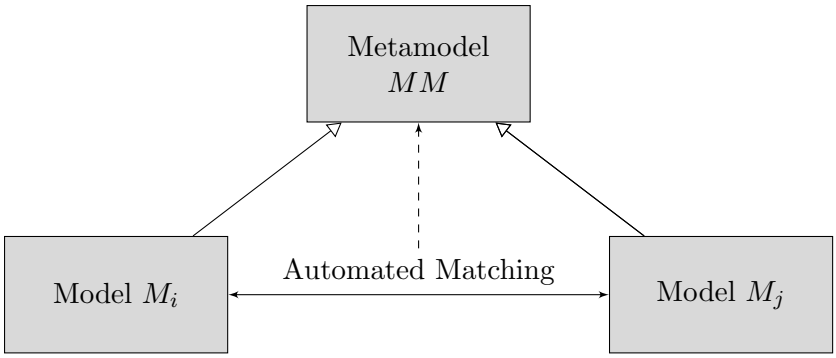

Figure 4.1.: Two models M_i and M_j conformal to the same metamodel MM. Based on the common metamodel, model matching can be performed automatically.

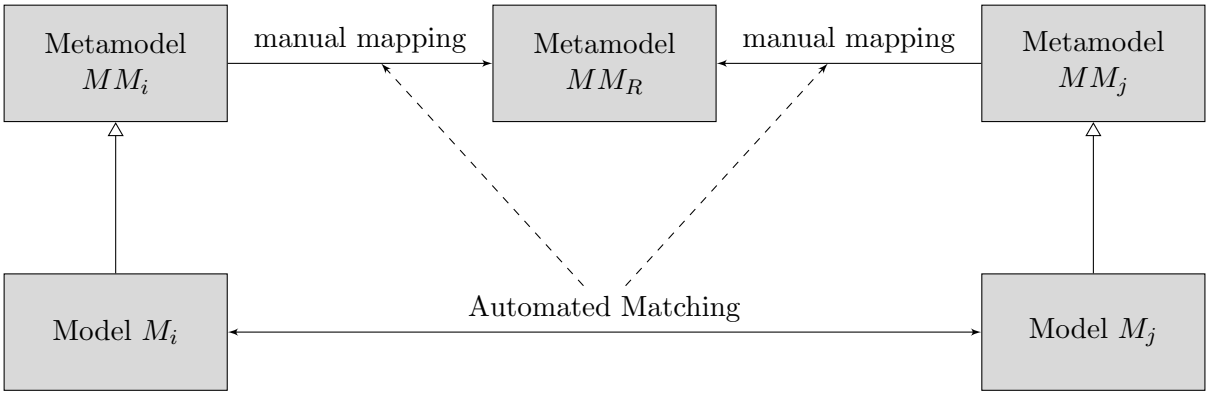

Figure 4.2.: Two models M_i and M_j conformal to different metamodels MM_i and MM_j. In order to match M_i and M_j their respective metamodel must be matched with a common reference metamodel MM_R in order to generate comparable object identifiers.

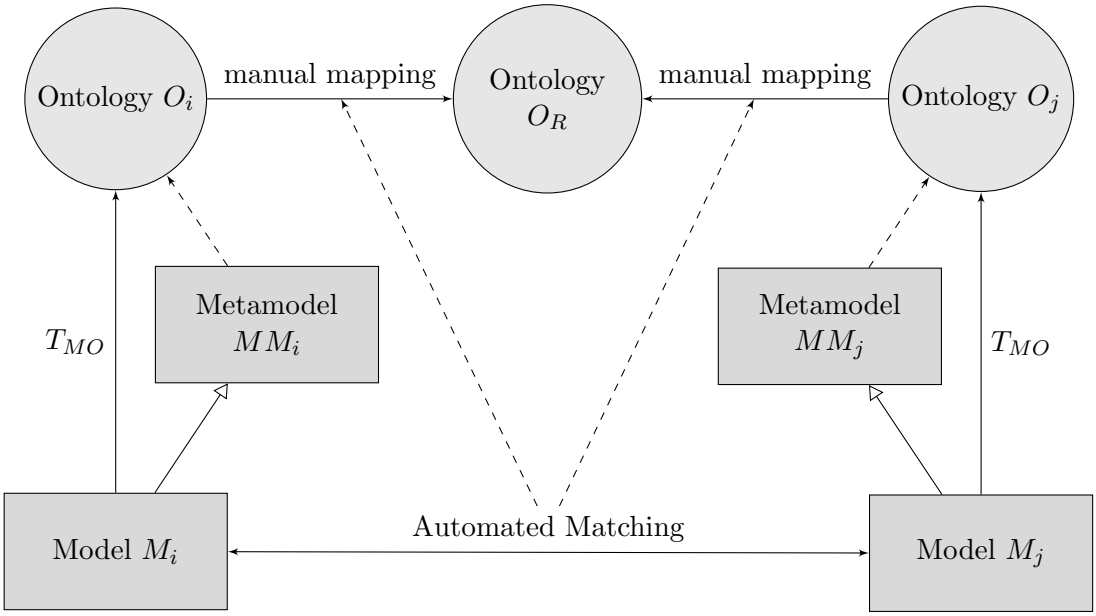

Figure 4.3.: The ontology-based model matching data flow is an excerpt of Figure 1.3. Model M_i and M_j are matched indirectly via the ontologies O_i and O_j derived from its respective model and mapping them manually to a common reference ontology O_R. The indirect matching is based on the assumption that the transformation T_{MO} is injective.

and objects representing an object which is not represented in the other model. Both results are a prerequisite for Merge Models Partially.

Figure 4.1 depicts a setup where two models have the same metamodel. In that case objects can be matched using the object classifiers specified by the shared metamodel as an equivalence criterion. As discussed in Chapter 2.1 a criterion based on a common classifier is not sufficient to match objects which have been created separately using a common metamodel, especially when the metamodel is very abstract from the application domain. Such a metamodel allows the Model Owner to use his own naming convention and to interpret the semantics of the metamodel elements differently.

For example, given that a metamodel MM provides a class `Composite`, an analysis of the objects in model M_i reveals that `Composite` consistently represents a physical component. Accordingly, M_i is structured in a system decomposition tree. Unlike model M_i, in model M_j a `Composite` object represents a physical Component or mission segment. A composition by physical components describes the spatial structure of the aircraft whereas a decomposition by mission segments describes the temporal structure of aircraft design missions. Now MM is extended by two classes `PhysicalComponent` and `MissionSegment` and the objects of both models are mapped to one of these new classes. In contrast to the mapping objects of M_i, the mapping of objects in M_j is ambiguous as every `Composite` object represents a `PhysicalComponent` or a `MissionSegment` object. As illustrated in Figure 4.2 the latter can be expressed by declaring each model conformal to a different metamodel and to map these metamodels to a common reference metamodel. Different interpretations can be formalized if the common reference metamodel provides not only the abstract concepts of the different metamodels which are suspected of beeing interpreted differently but also concrete concepts of the application domain. If the metamodels have been mapped to a reference metamodel, a common reference classifier can be part of a matching criterion. However, as metamodels do not provide a classifier for every domain concept, the objects still need to be matched by certain properties, usually the `name` property. If two model elements representing the same entity are created in different models, the modeler's strict adherence to the same naming convention is required in order to match objects by their name unambiguously. For instance, sophisticated and complex matching algorithms are required to determine the equivalence between an object `WingSpan` in one model and an object called `SpanMainWing` in another model. Even if that equivalence can be found, it is not clear whether the first object represents the span of the horizontal stabilizer whose context is completely different from the span of the main wing.

If the Model Owners map not only their metamodel but also the model to a common reference these mappings can be exploited to match models across inconsistent metamodel usage and different naming conventions. However, the process of manual model mapping is costly and additionally requires an agreement on the common reference. Furthermore, the closed world assumption of models can impose unsolvable integration problems. For instance, a UML metamodel specifies decomposition of its instance model by composition associations between classes. If two metamodels specify a different decomposition, an integrated object is contained in two different instances of disjoint classes which is an invalid construct in UML. Bridging semantic heterogeneity between models

using the syntactically identical metamodel has been described by Maalej [Maa10]. He also elaborated further that due to the open world assumption, ontologies particularly enable interoperability between different tools. In accordance with his considerations for work flow integration, the OIDA framework realizes the semantic matching of different conceptual aircraft models using an ontology as a common reference as depicted in Figure 4.3. Furthermore, the OIDA framework is designed to foster the commitment of the Model Owners to the ontology-based approach of integration. Therefore, OIDA supports Match Models involving the Model Owners in an interactive role.

Figure 4.4 shows the use case model of ontology-based model matching involving two actors, Model Owner and Ontology Owner. Furthermore, the Reasoner ensures the consistency of the ontologies which are created and modified within the Model Matching and the Ontology Engineering Application.

The Model Owner initiates the use case and is the primary actor of Find Equivalences, Review Equivalences, and Create Reference Concepts. Thereby, he is assisted by the Ontology Owner who is responsible for a consistent use of the equivalence statements among Model Owners.

The flow of events is described in detail in Figure 4.6 which is subsequently performed by all Model Owners involved in the conceptual design process. The resulting mapped ontologies establish an indirect semantic matching between the elements of the discipline specific models they are derived from. Thus, by performing Match Models with at least two discipline-specific models M_i the basis for the following Merge Models Partially is founded.

Reference Ontology Maintenance

In the Revise Reference Ontology use case which is performed in a separate Ontology Engineering System, the Ontology Owner improves the quality of the reference ontology O_R. During the matching process the Model Owners add new entities, such as individuals, classes and properties to the reference ontology O_R. After the Match Models the Ontology Owner revises O_R in order to maintain quality and reusability. Therefore, O_R comes under scrutiny especially regarding scope, style, context and redundancy of its contents.

Added concepts have to be within the **scope** of O_R. For instance, design tool-specific concepts are generally beyond the scope as O_R represents knowledge about aircraft design domain rather than knowledge about the use of design tools.

All concepts in O_R should conform to a **style guide** which not only addresses nomenclature and structural design patterns but also guidelines regarding the discrimination between instance and metaconcept level, e.g. whether wing is an individual or a class. The latter is of special importance for the correlation of models and their metamodels to ontologies [AK03]. The preliminary analysis of the sample models showed large differences between the respective naming conventions but no difference regarding the discrimination between instance and metaconcept level. Therefore, the naming convention of O_R is independent of the sample models, whereas the discrimination between instance and metaconcept is strongly oriented to the sample models. As a result O_R

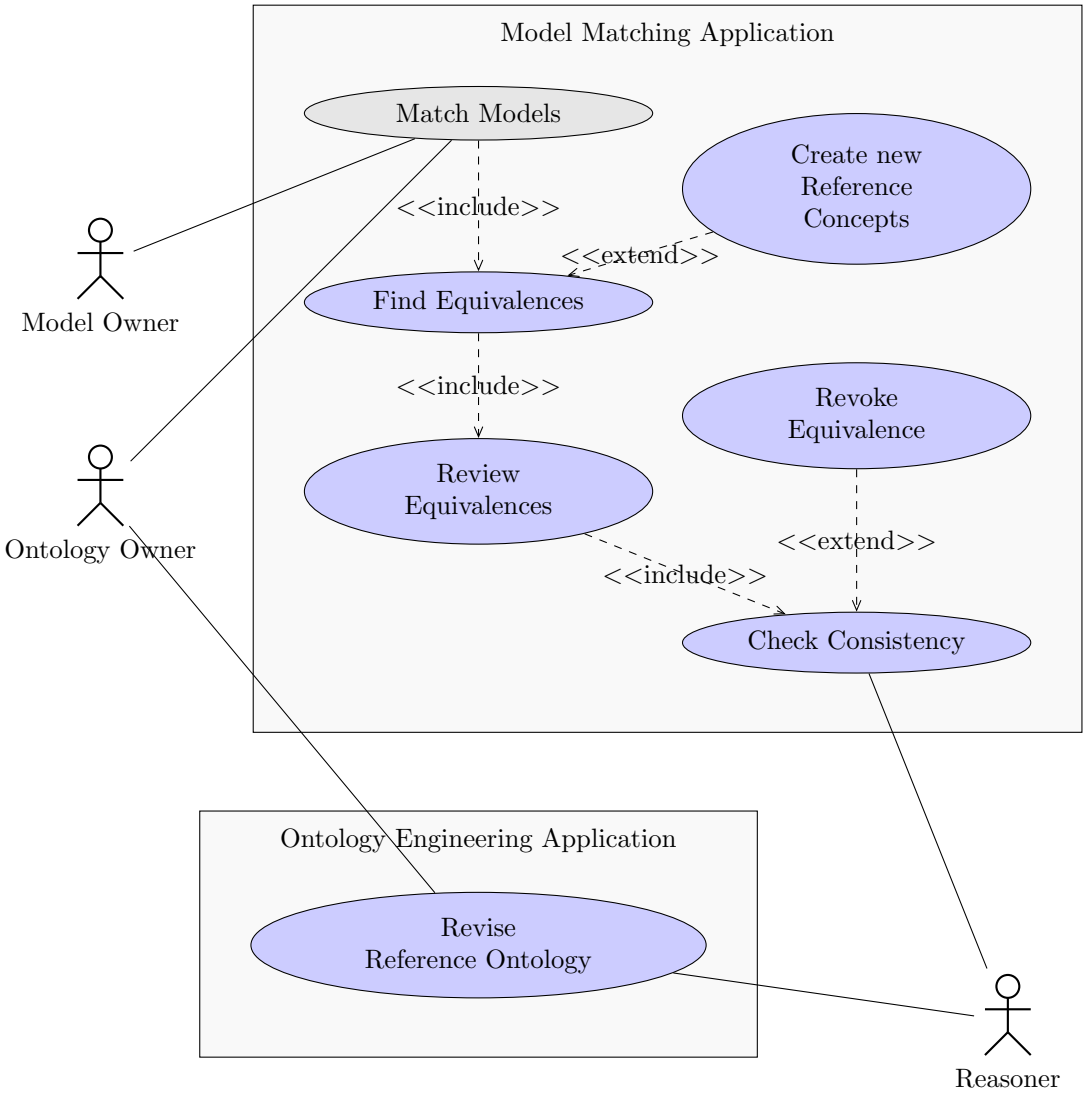

Figure 4.4.: Use case diagram of Match Models use cases

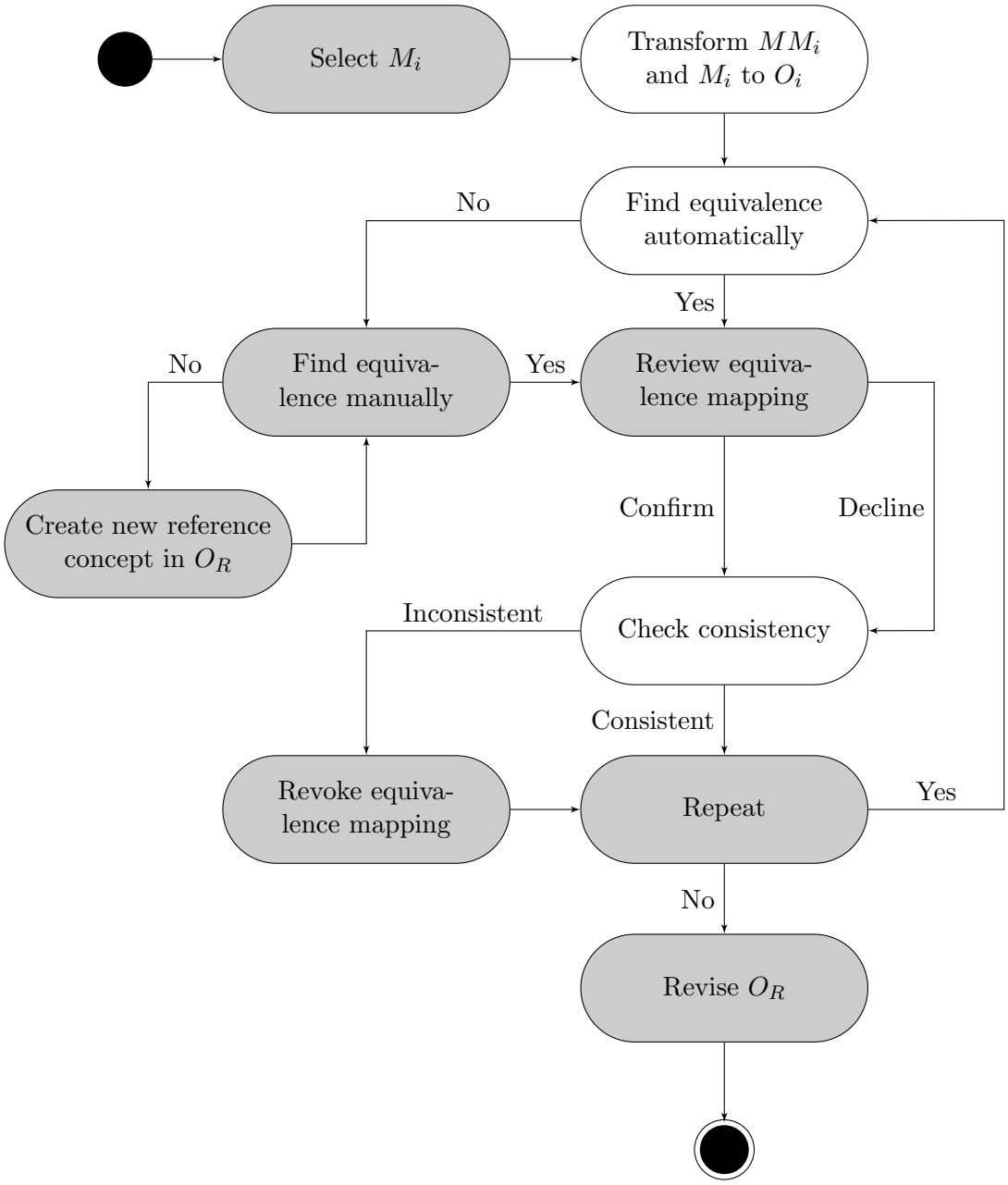

Figure 4.5.: The process of mapping an ontology O_i derived from a model M_i to a reference ontology O_R. The white-filled tasks are performed automatically whereas the gray-filled tasks are performed by the Model Owner optionally assisted by the Ontology Owner.

Name	Match Models
Participating actors	Initiated by Model Owner Communicates with Ontology Owner

Flow of events

1. The Model Owner selects the discipline-specific model M_i to be matched with the reference ontology O_R
2. OIDA transforms model M_i to ontology O_i and imports a predefined reference ontology O_R into O_i.
3. OIDA tries to find equivalences between classes, object properties, or datatype properties in O_i and O_R automatically using algorithms and heuristics.
4. If an equivalent reference cannot be found automatically for a particular concept in O_i, the Model Owner selects one in the reference ontology manually. For instance, in order to map a model element called `span` in the SIMCAD model, the Model Owner selects the reference individual `WingSpan` as a candidate for an equivalent concept. Thereby, the Ontology Owner assists the Model Owner in finding the appropriate reference concept as he is in general most familiar with O_R.
5. If the Model Owner cannot find an equivalent reference concept in O_R, not even with the help of the Ontology Owner, the Model Owner creates a new reference concept in O_R. Thereby, he receives advice from the Ontology Owner regarding decomposition and naming convention of O_R.
6. Every equivalence relation which has been found manually or automatically is reviewed by both actors and either confirmed or explicitly declined. For instance, OWL allows the declaration that two concepts are not equivalent which can prevent a false equivalence candidate being recommended again.

Figure 4.6.: Description of the Match Models use case

Flow of events	7. The ontology O_i with the new equivalence or nonequivalence relation to O_R is checked regarding consistency by the Reasoner. 8. If the Reasoner reveals an inconsistency, the new mapping is revoked. 9. Regardless of the reasoner result the actors decide whether to repeat the mapping process until the Model Owner has mapped all model elements relevant for integration to O_R. 10. If O_i is sufficiently mapped the Ontology Owner revises the reference ontology O_R.
Entry conditions	• The Model Owner selects a model M_i he is responsible for and wants to match with the reference ontology O_R
Exit conditions	• All model and metamodel elements of M_i which the Model Owner considers relevant for model integration are mapped via O_i to the reference ontology O_R. • O_R is revised by the Ontology Owner regarding structural integrity and reusability.
Quality requirements	• The Model Owner must comprehend the employed heuristics and algorithms in order be able to assess their results.

Figure 4.6.: Description of the Match Models use case (continued)

has far more individuals than classes. This decision is convenient for matching the given sample models but bears the risk of limited reusability of O_R to more domain-specific metamodels which could specify metaconcepts which are currently implemented as instances.

Ontology classes are interrelated by object properties and anonymous classes which can be further restricted by logical statements. Automated reasoners can evaluate these statements to classify concepts more specifically and to check the consistency of the overall ontology. Adding restrictions is a nontrivial maintenance task because it represents domain knowledge engineering which should be performed in collaboration with the Model Owner. Such restrictions should be added carefully for two reasons. First, it can be difficult for a Model Owner to discriminate between fundamental and project-specific constraints. For instance, most aircraft must have at least two independent engines. For civil aircraft this constraint stems from safety regulations. Military and unmanned applications may not be subject to this regulation. Accordingly, this constraint would limit O_R to civil transport aircraft applications. Therefore, the Ontology Owner should focus on object properties which are evaluated for the generation of matching recommendations as this will make the matching process more efficient.

During Model Matching the Model Owners and Ontology Owner can add synonyms to a concept which already exists. Generally, synonyms help the Model Owners and automated equivalence finders to determine reference concepts during Model Matching. For instance, `MaximumTakeOffWeight`, `MaximumTakeOffMass` and their respective abbreviations `MTOW` and `MTOM` are commonly used synonyms. In the reference ontology a **synonym relationship** must be explicitly declared by an object property. The Ontology Owner has to decide whether the synonym facilitates the automated matching through better results of a name equivalence finder or unnecessarily increases the complexity of O_R. However, the Model Owners may create reference synonyms which are not conformal to the reference ontology style guide or create inconsistency. For instance, O_R may contain a statement declaring `Mass` and `Weight` to be disjoint concepts. If `MaximumTakeOffWeight` and `MaximumTakeOffMass` have been classified as `Mass` and `Weight` respectively, a synonym relationship would lead to an inconsistent ontology.

New ontology classes and their instances have to be classified as in a clear and **intuitive taxonomy**. A clear taxonomy not only helps Model Owners and Ontology Owner manually navigating through ontologies. As classifications are also the main source of automated reasoners, they facilitate the detection of inconsistencies but also the automated inference of new knowledge.

If the Ontology Owner decides that a new reference concept is beyond the scope or not conformal to the style guide of O_R, he changes, deletes, or moves the concept to an external ontology. The existing equivalence mappings from ontologies derived from discipline-specific models have to be updated accordingly. Deletion, moving, or renaming of concepts should be performed by the Ontology Owner in collaboration with the affected Model Owners in order to foster the common understanding of the scope and the reference nomenclature.

As shown in Figure 4.4 the Revise Reference Ontology use case is performed in a dedicated Ontology Engineering Application. Accordingly, the use case is important for

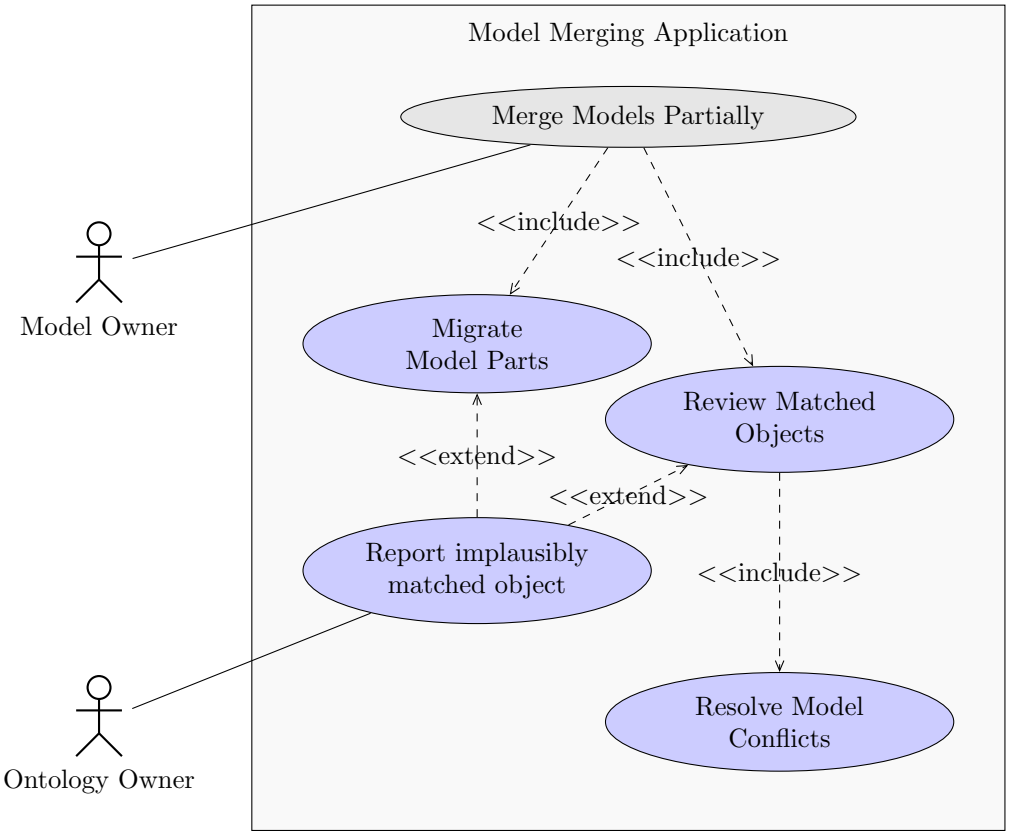

Figure 4.7.: Diagram of Merge Models Partially use cases

quality assurance of an essential artifact of the Integrate Models but is not addressed by the following analysis and design of the OIDA framework.

4.2. The Partial Model Merge Capability

The design of the Merge Models Partially use case described in this section is based on the assumption that at least two Model Owners have performed the Match Models use case. Furthermore, one of the models has been declared the Chief Engineering Model which overrules the other models in the case of conflict.

Merge Models Partially is the pivotal use case of ontology-based model integration as it realizes the Resolve Model Conflicts and Migrate Model Parts use cases described in Section 1.2. By performing the Match Models use case, the Model Owners created the basis for Merge Models Partially by mapping the concepts of their discipline-specific models to a common reference ontology.

Partial Model Merging includes Resolve Model Conflicts and Migrate Model Parts. These use cases can be performed by the Model Owner without assistance from the Ontology Owner within the `model integration system`.

The activity diagram Figure 4.8 illustrates the linear process of Merge Models Partially which is designed to give the Model Owner control over any changes applied during the resolution of semantic model conflicts and the migration of model elements. Furthermore, the Model Owner is enabled to report detected conflicts or elements proposed for migration which he considers implausible. The flow of events of Merge Models Partially is described in more detail in Figure 4.9.

4.3. The Co-Evolution Capability

After performing the Match Models and Merge Models Partially use cases, artifacts, such as discipline-specific models and ontologies, become coupled by equivalence mappings. As both use cases are usually performed during an active design process, these artifacts are subject to changes which are not automatically propagated to the other coupled artifacts. This section focuses on changes which affect the matching between discipline-specific models and the reference ontology. As these mappings are created in a semi-automated process, their efficient evolution is critical for the overall efficiency of Integrate Models.

Given an equivalence relationship between two entities, a change that does not affect the existence or the meaning of both entities leaves the equivalence relation intact. Otherwise, the equivalence relation has to be adjusted. Ideally, this adjustment can be performed automatically. However, it is a basic assumption of the Match Models use case that the equivalence mappings between the ontologies derived from the discipline-specific models and the reference ontology cannot be created fully automated but require a domain expert. Accordingly, if one end of an existing equivalence mapping has been changed, it cannot be adjusted automatically in every case.

As the concept of an ontology-based model integration process stipulates a frequent application, a high number of manual adjustments by the Model Owner would reduce the overall process efficiency significantly. The idea of co-evolution of coupled artifacts based on the hypothesis that if particular changes and their respective reason are recorded by a software system, it can efficiently propagate changes to coupled artifacts. Generally, a small evolutionary change requires a less complex change propagation and thus less manual clarifications. Therefore, how different kinds of reasons for change affect model integration related artifacts and how this knowledge can increase the applicability of automated adjustments of equivalence mappings was investigated. Four use cases depicted in Figure 4.10 can be discriminated. All of them extend the general *Co-Evolve Integration Artifacts* use case:

In *Evolve due to Aircraft Modeling* the Model Owner changes his model in an aircraft design process. The ontology which is derived from the discipline-specific model and its mappings to the reference ontology must be kept consistent. Metamodels of the

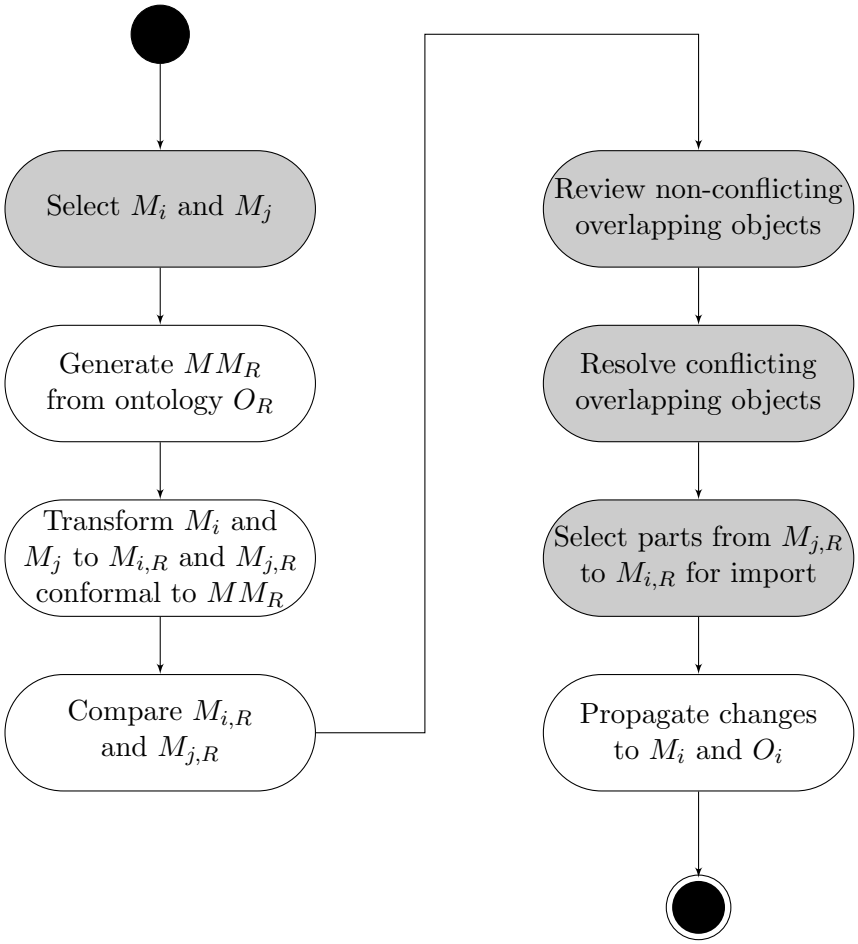

Figure 4.8.: The flow of events during the Merge Models Partially use case depicted in a UML activity diagram. The white-filled tasks are performed automatically whereas the gray-filled tasks are performed by the Model Owner optionally assisted by the Ontology Owner.

Name	Merge Models Partially
Participating actor	Initiated by Model Owner
Flow of events	1. The Model Owner selects his model M_i as the target model and the source model M_j he wants to perform Merge Models Partially with Merge Models Partially. Both models have been mapped to O_R. 2. A common metamodel MM_R is generated from O_R. 3. M_i and M_j are transformed to MM_R conformal $M_{i,R}$ and $M_{j,R}$. This transformation evaluates the equivalence mappings of the ontologies O_i and O_j to O_R. 4. $M_{i,R}$ and $M_{j,R}$ are compared determining non-conflicting matching objects, conflicting equivalent objects, and source model specific objects. The first two sets contain elements of the source model which are semantically matched to an equivalent element in the target model. The latter set of objects contain model elements which are only mapped to a reference concept but have no correspondence in the other model. 5. The Model Owner is presented a list of matched objects which have non-conflicting attribute values. 6. The Model Owner reviews this list of matched objects and reports implausible entries. 7. The Model Owner is presented a list of matched objects having conflicting attribute values. 8. The Model Owner reports implausible entries or decides whether to resolve the conflict by accepting the value from $M_{j,R}$. Assuming that he is not the owner of the source model M_j, the conflict cannot be resolved in the opposite direction.

Figure 4.9.: Description of the Merge Models Partially use case

Flow of events	7. The OIDA framework provides a list of objects in M_j which could not be matched with an object in M_i. These source model specific objects are presented according to the naming convention of O_R. Accordingly, the model owner is not exposed directly to M_j. The Model Owner selects objects for import and specifies an appropriate name and container object in the target model M_i according to his convention. 8. The conflict resolution decisions are carried out by propagating the respective attribute values from $M_{j,R}$ via $M_{i,R}$ to M_i using the $T_{M_i \to M_{i,R}}$. In order to keep O_i consistent with M_i equivalence mappings of the imported objects are adopted from O_j.
Entry conditions	• The Model Owner selects the model M_i which has been matched with O_R. • The Chief Engineering Model has been matched with the same O_R.
Exit conditions	• The conflict resolution and import decisions have been propagated to M_i • O_i is consistent with M_i • Review on implausible results are reported to the Ontology Owner • Assuming that both Model Owner and Ontology Owner are not responsible for M_j and O_j, both artifacts remain unmodified.
Quality requirements	• The Model Owner must comprehend the equivalence and conflict criterion in order to assess their effectiveness. • The Model Owner should not be directly exposed to the naming and decomposition convention of M_j

Figure 4.9.: Description of the Merge Models Partially use case (continued)

discipline-specific models can also be subject to change. However, the propagation of this kind of change to the instance model is beyond the scope of this dissertation.

In *Evolve due to Model Integration* models and ontologies are extended on two occasions. First, during the Match Models process new entities can be added to the reference. Second, during Migrate Model Parts new elements are added to the target model. After both processes models and ontologies must be kept consistent to each other. Accordingly, the Evolve due to Model Integration is included in the analysis of the Match Models and Migrate Model Parts. Therefore, the further analysis of Co-Evolve Integration Artifacts does not account for it.

The *Evolve due to Integration Software Maintenance* acknowledges that the implementation of the OIDA framework can be subject to change. Especially changes to the transformation algorithms have consequences for the artifacts of the ontology-based model integration process.

The modification of the reference ontology by the Ontology Owner is treated separately in the *Evolve due to Reference Ontology Maintenance* use case. The source ontologies which are mapped to the reference must be kept consistent with those changes.

Modifications on coupled artifacts can be characterized by their impact on topology and attributes. *Topology changes* affect relations of entities to each other and thus the overall structure of a set of entities. In that sense, the creation or the deletion of a new entity is also a topology change. *Attribute changes* are value modifications of a characteristic. The complexity of the consequences of an attribute change depends on the abstraction layer of the attribute. For example, changing the target of an aggregation relation in a metamodel changes the decomposition structure of objects in instance models. In contrast, changing a comma in a description attribute has considerably smaller implications.

The situation and required action to achieve consistency among co-evolving artifacts depends on the particular use case. In the following the situation and the respective required actions for topology and attribute changes are described in more detail.

Evolve due to Aircraft Modeling

During the aircraft design process, changes in discipline-specific models can affect the topology or can be limited to value changes of attributes. The latter kind of modification is more likely late in the design process when the design of the aircraft configuration has stabilized. The discipline-specific metamodels are usually not modified during an aircraft design process but more in the context of tool evolution which is not addressed in this dissertation.

Topology Changes Topology changes in models usually do not affect the semantics of the objects but the meaning of associations between objects. For instance, if an object `CargoBay` is moved from the container object `Fuselage` to the container object `PressurizedCompartments`, the former composition association changes the semantics from a component integral object composition to a classification inclusion relation. Accordingly, after this modification the semantic properties of the relation from `CargoBay` to

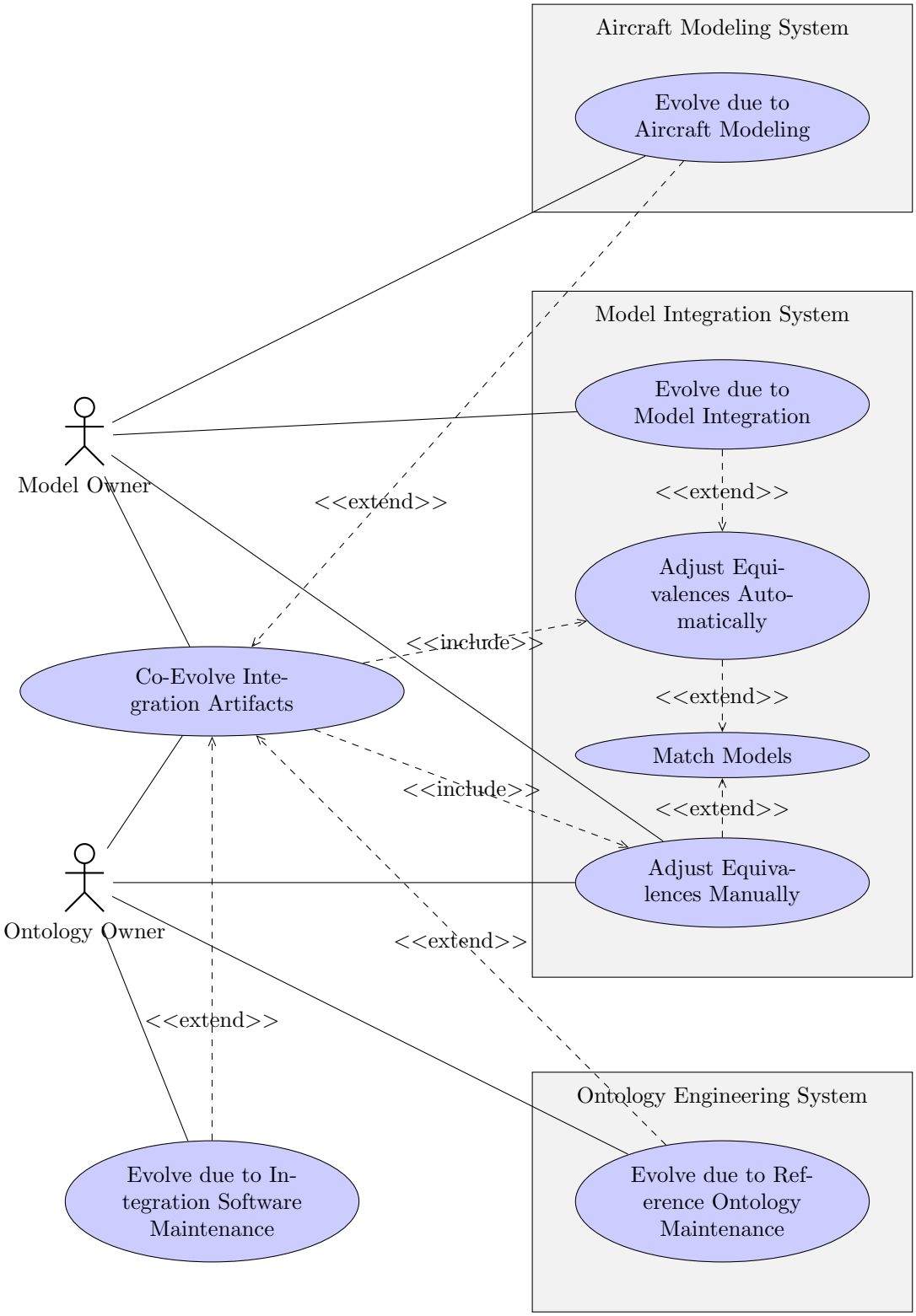

Figure 4.10.: Diagram of Co-Evolve Integration Artifacts use cases in the context of ontology-based model integration

its container are substantially different. However, regardless of whether the context of `CargoBay` is `Fuselage` or `PressurizedCompartments` its meaning remains unchanged and does not require manual review by the Model Owner.

Attribute Changes A value modification does not affect the semantics unless special attributes, such as the name, are changed. In that case, revision of existing equivalence mappings by the Model Owner is generally required. In general, attributes like concrete measured values do not constitute the semantics of an object and should not be mapped to a reference during the Match Models use case.

Evolve due to Integration Software Maintenance

This use case is focused on modifications to components of an OIDA implementation which affect artifacts involved in the model integration process and affect the ability to reuse the result of previous model matching sessions. In particular, modifications of transformation algorithms lead to changes in generated artifacts, such as derived ontologies.

Topology Changes Modifications of transformation algorithms can effectuate a different structure of the generated artifacts. The inheritance relation between previously matched associations of new relations can be exploited to adjust existing equivalence mappings.

Attribute Changes A modification of the transformation algorithms can affect attribute values in the same manner as in the Evolve due to Aircraft Modeling use case. The modification of the transformation regarding the processing of metamodels effectuates the processing of instance model elements. The OIDA can use refactoring methods to adjust existing equivalence mappings of instance concepts automatically. However, a review of these adjustments by the Model Owner is generally required.

Evolve due to Reference Ontology Maintenance

Maintenance modification of the reference ontology O_R are performed in an Ontology Engineering Application separated from the OIDA system. It potentially requires changes in all models which contain model elements that have been mapped to it.

Topology Changes In general, additive changes in the reference ontology do not require modifications of other artifacts. Additive changes can be (1) new classes to express the nature of concepts more discerningly, (2) new rules by anonymous classes in order to constrain the ontology, (3) new properties to add more explicit context to an ontology, and (4) individuals to cover more concrete instances of the classes.

However, at least the review of adjustments by the Model Owner is required if the scope of the reference ontology is changed. For instance, the Ontology Owner may decide to exclude certain concepts from the reference ontology in order to narrow the

scope without replacing it by a concept in an external ontology. Or the Ontology Owner introduces a new concept which is incompatible with an existing one. Furthermore, the Ontology Owner can transform a class to an individual and vice versa.

Attribute Changes In general, value modifications to an entity do not require modifications to the ontologies mapped to the reference ontology. Only renaming a reference concept requires a migration of the mappings to the new identifier which have to be re-validated by the Model Owner in the same manner as model attribute changes in the Evolve due to Reference Ontology Maintenance use case.

As the Evolve due to Aircraft Modeling is governed by the Match Models and Merge Models Partially use cases, it is treated as an integral part of both use cases. The other use cases of Co-Evolve Integration Artifacts due to modifications outside the OIDA framework such as Evolve due to Aircraft Modeling, Evolve due to Integration Software Maintenance, and Evolve due to Reference Ontology Maintenance can be covered by the Adjust Equivalences Automatically Adjust Equivalences Manually which is focused on importing previous ontology mappings to a generated ontology. Both can be combined in the flow of events illustrated in Figure 4.11. This generic co-evolution process is described in Figure 4.12.

The heuristics and algorithms employed in the Adjust Equivalences Automatically use case and the generation of adjustment recommendations for the Adjust Equivalence Manually use case must be transparent to the Model Owner in order to convey acceptance of the co-evolution capabilities of the OIDA framework in particular and of the sustainability of the model integration process in general. Evolutionary changes on artifacts and framework algorithms imply a continuous relation between the old and the newly evolved state. A shared understanding of the adjustment heuristics by Model Owner and Ontology Owner help them to initiate the Co-Evolve Integration Artifacts use case in time, before modifications become too radical for any automated adjustment.

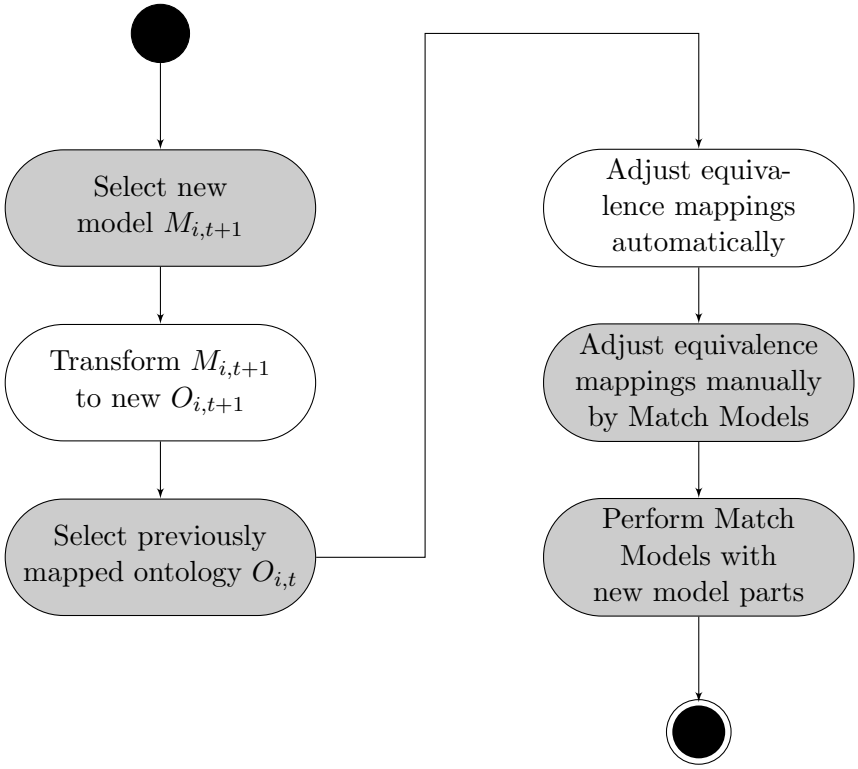

Figure 4.11.: The flow of events during the Co-Evolve Integration Artifacts use case addressing the use cases Evolve due to Aircraft Modeling, Evolve due to Integration Software Maintenance, and Evolve due to Reference Ontology Maintenance. The white-filled tasks are performed automatically, whereas the gray-filled tasks are performed by the Model Owner assisted by the Ontology Owner.

Name	Co-Evolve Integration Artifacts
Participating actors	Initiated by Model Owner Communicates with Ontology Owner
Flow of events	1. The Model Owner selects a new model $M_{i,t+1}$. 2. The Model Owner selects ontology $O_{i,t}$ which contains the previously created equivalence mappings. 3. $M_{i,t+1}$ is transformed to a new ontology $O_{i,t+1}$ by the new transformation $T_{MO,t+1}$. 4. OIDA adjusts both ends of every equivalence mapping to $M_{i,t+1}$ and $O_{R,t+1}$. If one of the ends of the mapping cannot be found or if there is more than one solution, the mapping is declared obsolete and put on a list for later manual adjustment. 5. The Model Owner reviews the obsolete mappings and either decides to discard them or adjusts them manually. 6. The Model Owner performs the Match Models use case as described in Figure 4.6.
Entry conditions	• $M_{i,t}$, which is a previous version of the new discipline-specific model $M_{i,t+1}$, has been mapped to the old reference ontology $O_{R,t}$ in a Match Models use case declaring equivalence statements in $O_{i,t}$.
Exit conditions	• Previously created equivalence mappings have been readjusted to the new versions of $M_{i,t+1}$ and OIDA framework.
Quality requirements	• Most of the existing equivalence mappings could be readjusted automatically. • The Model Owner comprehends the automated adjustments made by OIDA.

Figure 4.12.: Detailed description of the Co-Evolve Integration Artifacts M_i, O_i, and O_R which have been modified between the points in time t and $t+1$. This use case addresses Evolve due to Aircraft Modeling, Evolve due to Integration Software Maintenance, and Evolve due to Reference Ontology Maintenance

5. The Oida Analysis Object Model

> "[...] because in the age of information overload the ultimate luxury is meaning and context."
>
> *Wired premiere issue*

An analysis object model defines and formalizes participating objects and their relations to each other. The objects are categorized in Entity Objects, Boundary Objects, and Control Objects as defined by Jacobson et al. [JBR99]. *Entity Objects* represent information objects which are created or read from external resources, manipulated and linked with other objects, and destroyed or written to external resources by software system. *Boundary Objects* enable the communication of the system with other external systems or actors. *Control Objects* are concerned with effective and efficient performance of system components during use cases and their interaction with the users through Boundary Objects. In the following chapter, this analysis is carried out to cover the participating objects of the Match Models, Merge Models Partially, and Co-Evolve Integration Artifacts use cases. Additionally, this chapter also elaborates the analysis of the ontology-based model transformation T_M as an automated operation.

5.1. Analysis Objects of the Match Models Use Case

The Match Models use case enables the matching of semantically equivalent elements of different discipline-specific models by mapping the discipline-specific concepts extracted from the models to concepts in a reference ontology.

The Oida facilitates the semi-automated mapping process by proposing automatically identified equivalences and by supporting the Model Owner in creating equivalence relations and in reviewing the relations manually. Thereby, the Ontology Owner being more familiar with the capabilities of the Oida and the reference ontology moderates the process and guides the Model Owner's actions in a consultative manner.

In the following, the participating objects of the Match Models use case are identified. The use case is described in Section 4.1.

Figure 5.1 shows Entity Objects and their associations with each other. The use case is initiated by the Model Owner who selects the `DisciplineSpecificModel` he owns. This action triggers the generation of the `DerivedOntology` by the `Ontology-Generator`. The `EquvivalenceFinder` imports the `ReferenceOntology` and starts creating `EquivalenceCandidates` if it finds semantically equivalent `ReferenceConcepts` for given `DisciplineSpecificConcepts`. If `EquivalenceCandidates` cannot be found

automatically, the Model Owner creates them manually, if necessary by creating new `ReferenceConcepts` in the `ReferenceOntology`. The `Model Owner` reviews all `EquivalenceCandidates` by confirming, declining, or discarding them. The review is tentatively carried out in the reference ontology by creating equivalence statements specified by confirmed `EquivalenceCandidates` and nonequivalence relations specified by declined `EquivalenceCandidates`. The ontology extended by these new statements is checked for consistency using the automated `Reasoner`. Newly created statements which cause inconsistency are rolled back. The semi-automated semantic mapping between `DerivedOntology` and `ReferenceOntology` is repeated until the `Model Owner` has mapped all objects in scope of the reference ontology. Thereby, the Ontology Owner assists the `Model Owner` in manually finding `EquivalenceCandidates` and creating new `ReferenceConcepts` according to the style guide of the `ReferenceOntology`. The `Ontology Owner` uses his deeper understanding of the OIDA capabilities and the `ReferenceOntology`, and his experience with other `Model Owners` to convey a consistent use of the OIDA framework and a sustainable evolution of the `ReferenceOntology`. In a conciliatory manner, the `Ontology Owner` should not only guide the `Model Owner` but also communicate the rationale of the OIDA framework in general and heuristics of the `EquivalenceFinder` in particular. A list of the identified Entity Objects is given in Table 5.1.

The identified Boundary Objects and their relation to the Entity Objects of the Match Models use case are depicted in Figure 5.2. The `Model Owner` uses a `ModelNavigator` to select the `DisciplineSpecificModel` he owns. For the manual creation of `EquivalenceCandidates`, the `Model Owner` uses the `OntologyNavigator` to navigate the `DerivedOntology` and `ReferenceOntology`. The navigators must display the available models and ontologies respectively in a representation which is familiar to the `Model Owner`. The `EquivalenceReviewForm` provides the `Model Owner` with automatically and manually found `EquivalenceCandidates` for review. A list of these Boundary Objects in Table 5.2 describes them in more detail.

The Match Models use case is controlled by `MatchingControl` depicted in Figure 5.3. It triggers and records the creation of equivalence and nonequivalence relationships in order to roll back the creation if the `Reasoner` determines an inconsistency in the ontologies. A definition of the identified Control Object is given in Table 5.3.

5.2. Analysis Objects of the Ontology-based Model Transformation T_M

The ontology-based model transformation is an operation associated with the Match Models use case as it realizes the essential capability of the OIDA framework to match objects from `DisciplineSpecificModels` which are neither conformal to the same metamodel nor have the same convention for naming and model structure. However, it is not performed immediately after a Match Models session but immediately before the Merge Models Partially use case in order to use the latest semantic mappings created by the Model Owner. Generally, the particular algorithm of this transformation is not

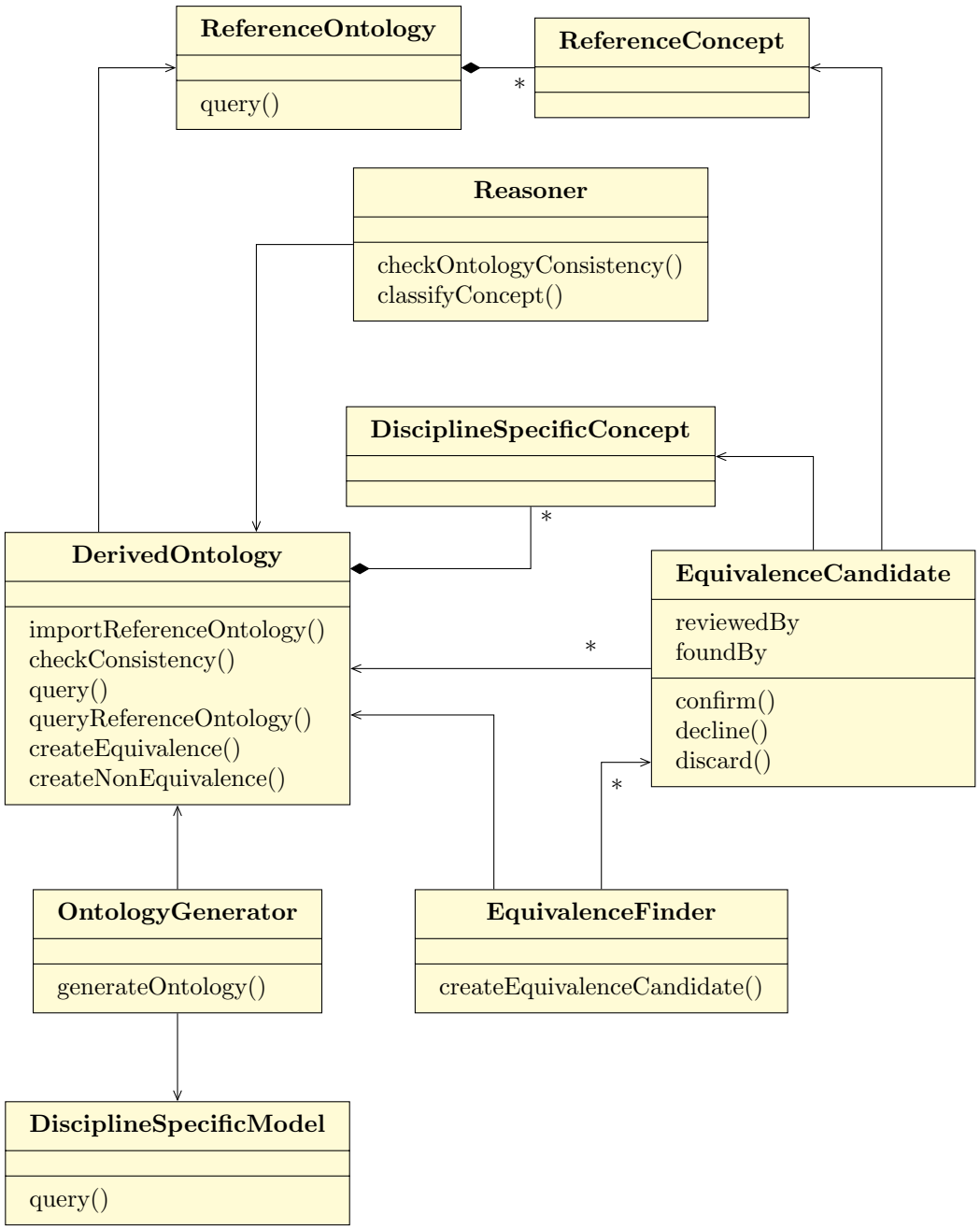

Figure 5.1.: Diagram of Entity Objects of the Match Models use case

Entity Object	Definition
DisciplineSpecific-Model	The DisciplineSpecificModel is owned by the Model Owner. It is a MOF conformal conceptual aircraft model which has a specific scope and focus on the overall aircraft.
OntologyGenerator	Generator which extracts concepts from a given DisciplineSpecificModel to create a DerivedOntology
DerivedOntology	This ontology represents the concepts of the DisciplineSpecificModel, such as ontology classes, object properties, datatype properties, and individuals. It contains equivalence or nonequivalence relations to the ReferenceOntology which are tagged with the creator and reviewer of the EquivalenceCandidate they have been created from.
DisciplineSpecific-Concept	An ontology representation of a semantically relevant element of the DisciplineSpecificModel.
ReferenceOntology	An ontology which contains a formal representation of aircraft design specific concepts in a reference naming convention.
ReferenceConcept	Different kinds of concept representations in the reference ontology, such as class, object property, datatype property or individual.
EquivalenceFinder	Automatically finds semantically equivalent ReferenceConcepts for given DisciplineSpecificConcepts and creates EquivalenceCandidates
EquivalenceCandidate	Holds a reference to a DiscriplineSpecificConcept and a semantically equivalent ReferenceConcept. It is reviewed by the Model Owner. Confirming the EquivalenceCandidate triggers the creation of an equivalence relation in the DerivedOntology, declining triggers the creation of a nonequivalence relation in the DerivedOntology, discarding triggers the deletion of the EquivalenceCandidate. It records its creator and reviewer.
Reasoner	The automated Reasoner checks the ontologies for consistency or classifies particular concepts.

Table 5.1.: Definitions of Entity Objects of the Match Models use case

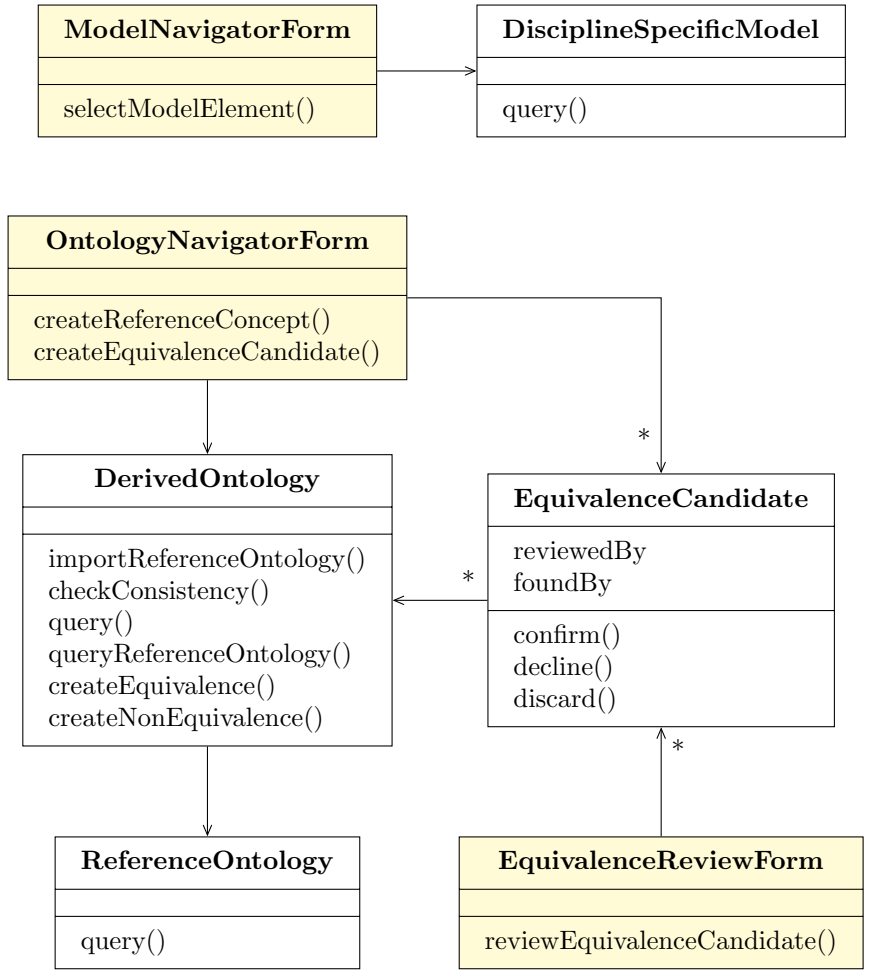

Figure 5.2.: Diagram of Boundary Objects of the Match Models use case

Boundary Object	Definition
`ModelNavigatorForm`	Displays available MOF conformal `DisciplineSpecificModels` and allows the Model Owner to select a model he wants to match.
`OntologyNavigatorForm`	Supports the Model Owner to manually find `EquivalenceCandidates` by presenting `DisciplineSpecificConcepts` and `ReferenceConcepts` in a structure which is clear and intuitive.
`EquivalenceReviewForm`	Supports the Model Owner in the review of `EquivalenceCandidates`

Table 5.2.: Definitions of Boundary Objects of the Match Models use case

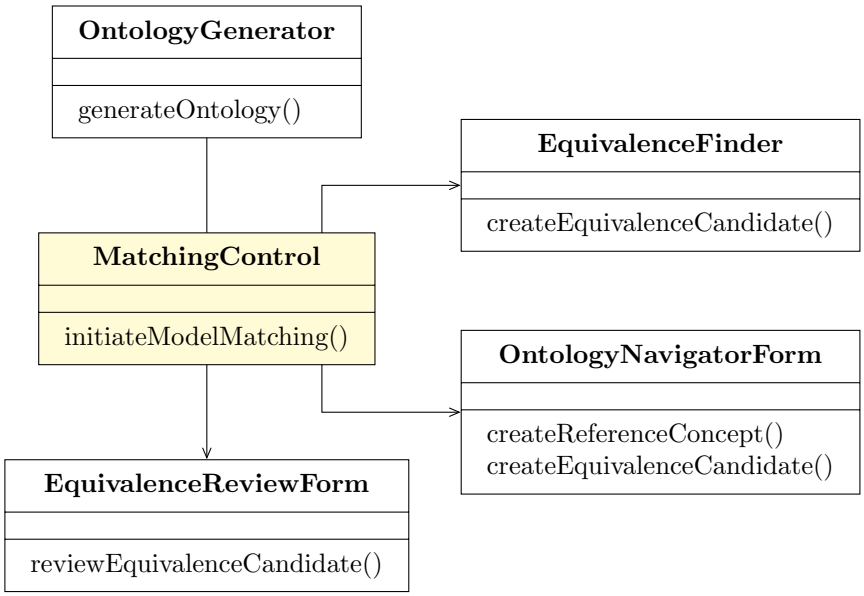

Figure 5.3.: Diagram of the MatchingControl object of the Match Models use case and its relations to other analysis objects

Control Object	Definition
`MatchingControl`	Manages the flow of events in the Match Models use case. It triggers the generation of the `DerivedOntology`, the import of the `ReferenceOntology` and commences the iterative semi-automated creation of equivalence relationships from `DerivedOntology` to the `ReferenceOntology`.

Table 5.3.: Definition of the `MatchingControl` object of the Match Models use case

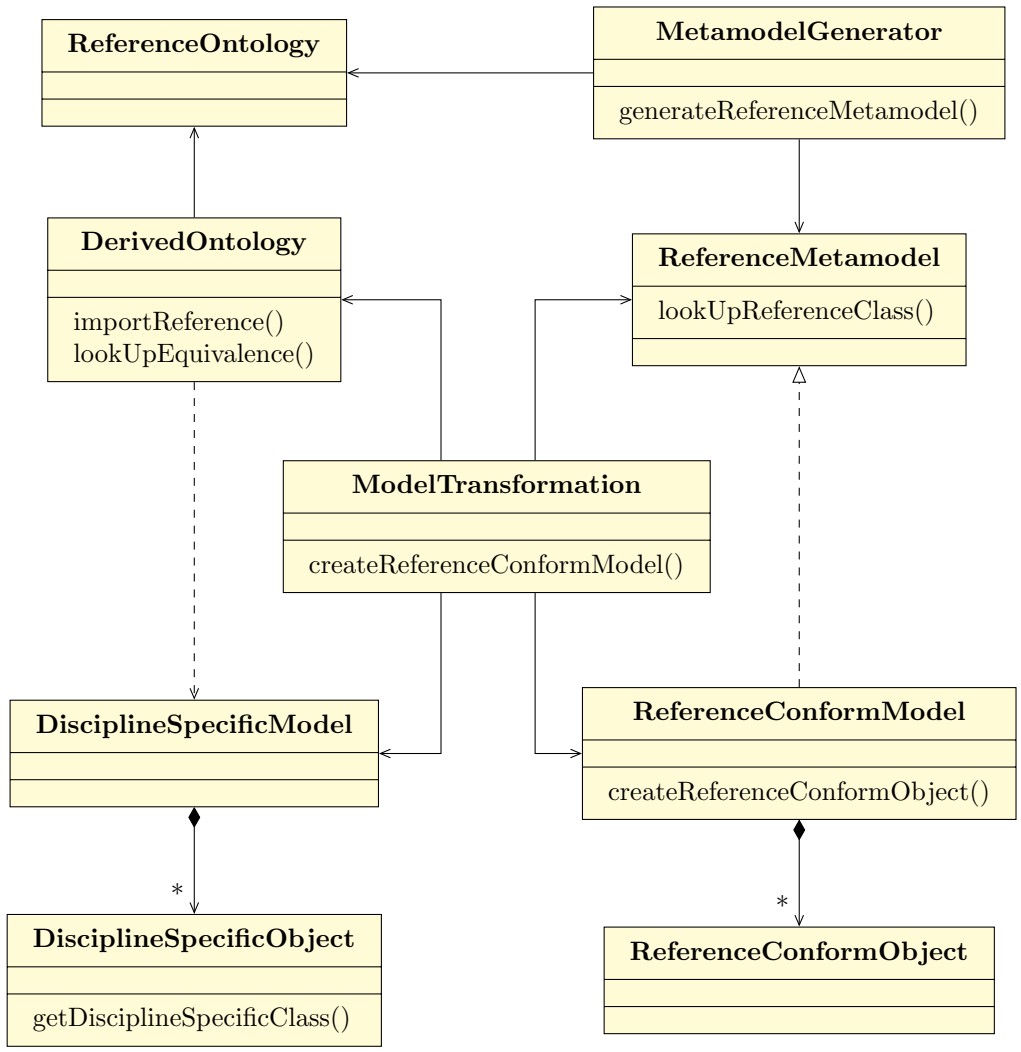

Figure 5.4.: Diagram of Entity Objects of the ontology-based model transformation T_M

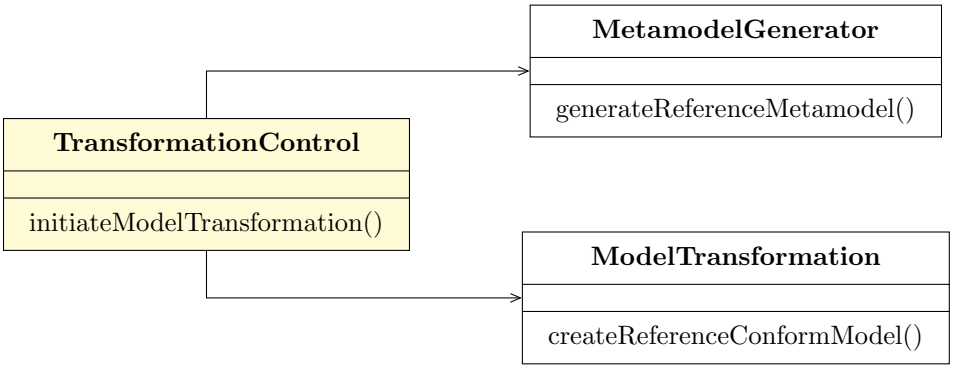

Figure 5.5.: Diagram of Control Object of the ontology-based model tansformation T_M

relevant for the Model Owner whereas the Ontology Owner must understand it in order to moderate the Match Models in a way that the equivalence relations can be well processed and to be able to assess the result. The following object analysis model of the ontology-based model transformation describes how the OIDA framework uses the result of one Match Models and prepares the Merge Models Partially use case.

Table 5.4 gives an overview of the aforementioned Entity Objects of the ontology-based model transformation. At first the `MetamodelGenerator` translates the metaconcepts of the `ReferenceOntology` to the MOF conformal `ReferenceMetamodel`. This metamodel is the target metamodel which provides classes in the type system of the OIDA runtime environment. Then the `ModelTransformation` is triggered to start creating the `ReferenceConformModel`. Thereby, for each `DisciplineSpecificObject` in the `DisciplineSpecificModel`, the `ModelTransformation` queries the `DerivedOntology` for the appropriate equivalence relation to the `ReferenceOntology`. If it can find an equivalent reference concept for the `DisciplineSpecificObject`, the `ModelTransformation` looks up the appropriate target reference class and transforms the `DisciplineSpecificObject` to the found reference class. By the transformation of all `DisciplineSpecificObjects` which have been mapped to the `ReferenceOntology` the `ReferenceConformModel` is created.

The process is controlled by the `TransformationControl` defined in Table 5.5 which triggers the `MetamodelGenerator` and the `ModelTransformation`. Its relation to other Entity Objects is depicted in Figure 5.5

By this ontology-based model transformation, all models which have been matched with the `ReferenceOntology` can be translated into a common metamodel which allows use of off-the-shelf model comparison and merging technologies.

5.3. Analysis Objects of Merge Models Partially Use Case

During the Merge Models Partially use case described in Section 4.2 the OIDA framework enables the Model Owner to resolve conflicts between semantically overlapping models

Entity Object	Definition
`MetamodelGenerator`	Translates the `ReferenceOntology` into the type system of the runtime environment of the OIDA framework generating a `ReferenceMetamodel`.
`ReferenceOntology`	See Table 5.1
`ReferenceMetamodel`	MOF conformal metamodel generated from the `ReferenceOntology` providing the concepts of the reference ontology in the type system of the Merge Models Partially runtime environment.
`ModelTransformation`	Transforms all objects of a `DisciplineSpecificModel` to objects in the `ReferenceConformModel`. Thereby, for the creation of a new `ReferenceObject` as instance of the equivalent class of the `ReferenceMetamodel`, it queries the `DerivedOntology` for the equivalence mapping of each `DisciplineSpecificObject` to the `ReferenceOntology`.
`DisciplineSpecificModel`	See Table 5.1.
`DerivedOntology`	See Table 5.1.
`ReferenceConformModel`	A model which conforms to the `ReferenceMetamodel` and has been generated from the `DisciplineSpecificModel` by `ModelTransformation`.

Table 5.4.: Definitions of Entity Objects of the ontology-based transformation of models to a common reference metamodel

Control Object	Definition
`TransformationControl`	Manages the ontology-based model transformation by triggering the `MetamodelGenerator` and the `ModelTransformation`. Thereby, it keeps track of the mapping between `DisciplineSpecificObjects` and `ReferenceConformObjects`.

Table 5.5.: Definition of the Control Object of ontology-based transformation of models to a common reference metamodel

and to import model elements from a discipline-specific source model to a discipline-specific target model. Figure 5.6 shows the participating Entity Objects of this use case.

The `Model Owner` selects the `DisciplineSpecificSourceModel` as source and the `DiscriplineSpecificTargetModel` as target of the Merge Models Partially use case. This triggers the ontology-based model transformation of both models to the `ReferenceConformSourceModel` and `ReferenceConformTargetModel`, respectively, conformal to the same `ReferenceMetamodel`. When these transformations have been completed, the `ModelComparator` tries to match the `ReferenceConformSourceObjects` to equivalent `ReferenceConformTargetObjects`. For every pair of matched objects, the `ModelComparator` creates `SemanticMatching` or a `SemanticConflict` if the matched objects are in a conflicting state. If a `ReferenceConformSourceObject` cannot be matched, the `ModelComparator` creates an `ImportCandidate` which references the `ReferenceConformSourceObject` which has no apparent semantic correspondence in the `ReferenceConformTargetModel`.

The `Model Owner` reviews instances of `SemanticMatching`, `SemanticConflict`, and `ImportCandidate`. The Model Owner can tag any of them for later fixing if he considers them to be implausible. If he confirms a `SemanticMatching` he indicates that he considers it to b plausible. By confirming a `SemanticConflict` he accepts adoption of the state of the `ReferenceConformSourceObject` to the `ReferenceConformTargetObject`. The OIDA framework does not allow resolution of a conflict by an adoption in the opposite direction as it assumes that the `Model Owner` only has authority over the `DisciplineSpecificTargetModel`. If an `ImportCandidate` is confirmed, it needs to be prepared for the import to the `DisciplineSpecificTargetModel`. For this purpose, the `Model Owner` assigns an appropriate name for the imported Object and a container Object both adhering to the naming and decomposition convention of the `DisciplineSpecificTargetModel` he owns.

When the `Model Owner` has completed the review, the `ChangePropagator` is triggered to carry out the required changes. For every confirmed `SemanticConflict` it copies the state from the referenced `ReferenceConformSourceObject` to the `ReferenceConformTargetObject` and changes the corresponding `DisciplineConformSourceObject`. For every confirmed `ImportCandidate`, it copies the `ReferenceConformSourceObject` to the `ReferenceConformTargetModel` creating a `SemanticMatching`. Then it creates an appropriate `DisciplineSpecificTargetObject` in the `DisciplineSpecificTargetModel` using the ontology-based `ModelTansformation` in the opposite direction. The change propagation is concluded by addition of a newly derived concept to the `DerivedOntology` and an equivalence mapping to the `ReferenceConcept` which is derived from the identifier of the `ReferenceConformTargetObject`. Thereby, the `ChangePropagator` not only establishes consistency between semantically equivalent objects from two different `DomainSpecificModels` but also updates the `DerivedOntology`.

During the Merge Models Partially use case, the Model Owner uses the Boundary Objects depicted in Figure 5.7. The `SemanticReviewForm` supports the Model Owner in confirming or reporting `SemanticMatchings`, `SemanticConflicts` and `ImportCandidates`. Concerning the latter, the `ImportCandidateCompletionForm` supports the

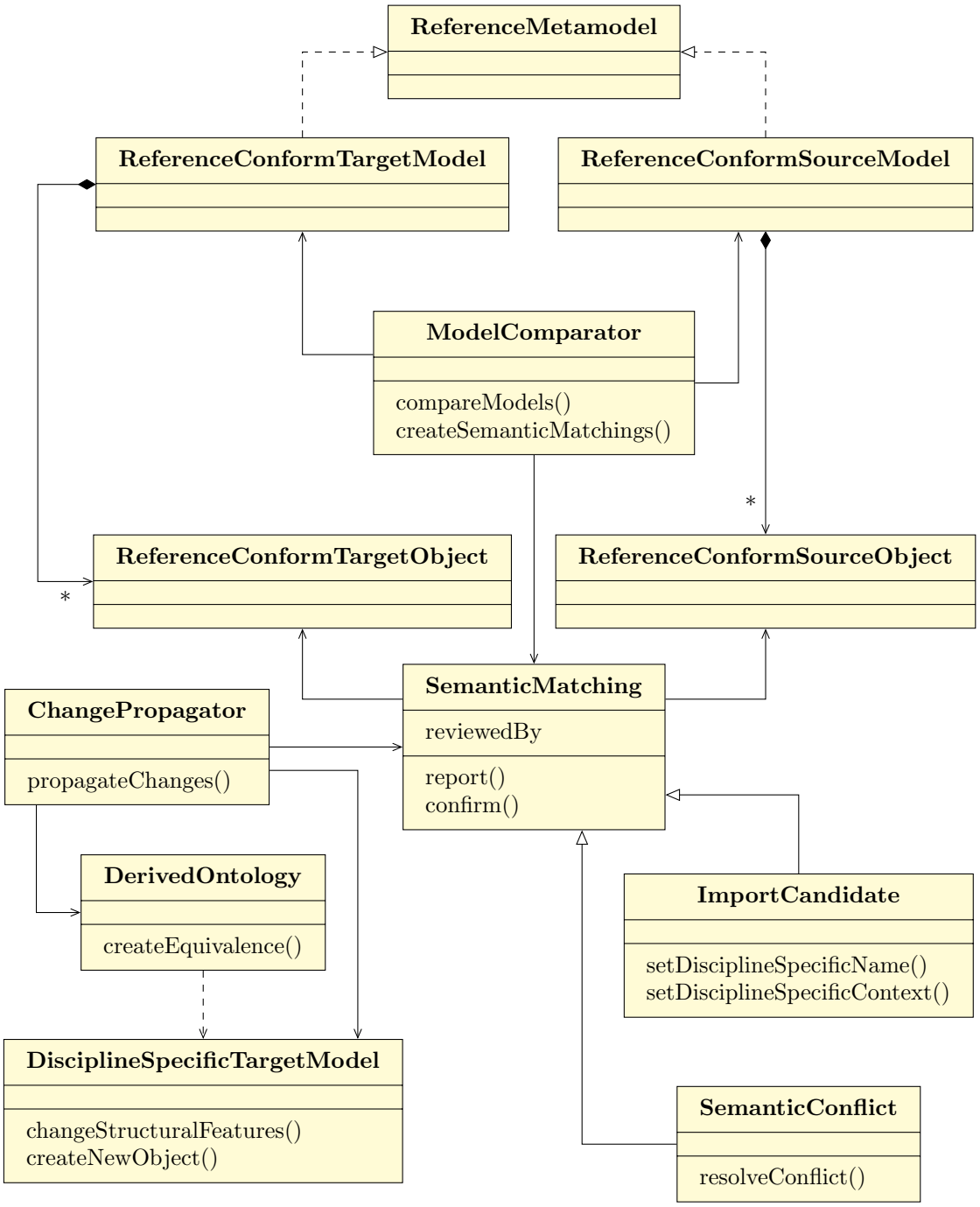

Figure 5.6.: Diagram of Entity Objects of the Merge Models Partially use case

Entity Object	Definition
ChiefEngineering-Model	A kind of DisciplineSpecificModel which is owned by the Chief Engineer who is entitled to finally decide on design conflicts.
LegacyModel	A kind of DisciplineSpecificModel which has been developed in preceding aircraft design projects.
DisciplineSpecific-SourceModel	A kind of DisciplineSpecificModel which is selected by the Model Owner for Merge Models Partially. It can be a Legacy Model or the Chief Engineering Model. The Model Owner is usually not entitled to modify it.
ReferenceMetamodel	See Table 5.4
ReferenceConform-SourceModel	A kind of ReferenceConformModel which has been transformed from the DisciplineSpecificTargetModel. The Model Owner wants to integrate his DisciplineSpecific-TargetModel with it as it is usually the Chief Engineering Model
ReferenceConform-SourceObject	A kind of ReferenceConformObject which has been transformed from the DisciplineSpecificSourceModel
ReferenceConform-TargetModel	A kind of ReferenceConformModel which is owned by the Model Owner. This model is the target of the ReferenceConformSourceObjects which the Model Owner wants to import. It contains ReferenceConformTargetObjects.
ReferenceConform-TargetObject	A kind of ReferenceConformObject which has been transformed from the DisciplineSpecificTargetModel.
ModelComparator	Compares two models conformal to the same ReferenceMetamodel creating SemanticMatchings between equivalent ReferenceConformSourceObjects and ReferenceConformTargetObjects or a SemanticConflict if the objects are in a conflicting state. It creates ImportCandidates from the ReferenceConformSourceObjects it cannot match.
SemanticConflict	A kind of SemanticMapping between two objects in a conflicting state. The conflict can be resolved by adopting the state of the source object.
ImportCandidate	Reference to an unmatched ReferenceConformSourceObject which can be imported into the DisciplineSpecificTarget-Model if the Model Owner specifies a name and container for the object, conformal to the naming and structure convention of the target model.

Table 5.6.: Definitions of Entity Objects of the Merge Models Partially use case

Entity Object	Definition
ChangePropagator	Propagates the conflict resolution and import decisions of the Model Owner to his DisciplineSpecificTargetModel and its latest DerivedOntology.
DisciplineSpecific-TargetModel	A kind of DisciplineSpecificModel owned by the Model Owner. In this model, the ChangePropagation changes the state of existing objects to resolve conflicts with the source model and creates new objects which have been imported from the source model.
DerivedOntology	The ontology derived from the DisciplineSpecificTarget-Model is a target of ChangePropagation which creates new derived concepts and maps them to the ReferenceOntology (see Table 5.1).

Table 5.6.: Definitions of Entity Objects of the Merge Models Partially use case (continued)

Model Owner in specifying a domain-specific name and container object for the respective ImportCandidate. These Boundary Objects are defined in Table 5.2 in more detail.

The Merge Models Partially use case is controlled by PartialMergingControl defined in Table 5.8. This Control Object and its relation to other Analysis Objects are depicted in Figure 5.8. It starts the Ontology-based Model Transformation after the Model Owner has selected the source and the target model of the current Merge Models Partially use case. At the end of the use case when the Model Owner has completed the review of SemanticMatchings and SemanticConflicts as well as the selection and preparation of the ImportCandidates, it starts the ChangePropagator.

Boundary Object	Definition
SemanticMatchingReviewForm	Supports the Model Owner in reviewing and deciding on conflicting SemanticMatchings and SemanticConflicts.
ImportCandidateCompletionForm	Supports the Model Owner in selecting and completing ImportCandidates. Thereby, it contains a ModelNavigator which allows selection of new existing or creation of new container objects in the DisciplineSpecificTargetModel.

Table 5.7.: Definitions of Boundary Objects of the Merge Models Partially use case

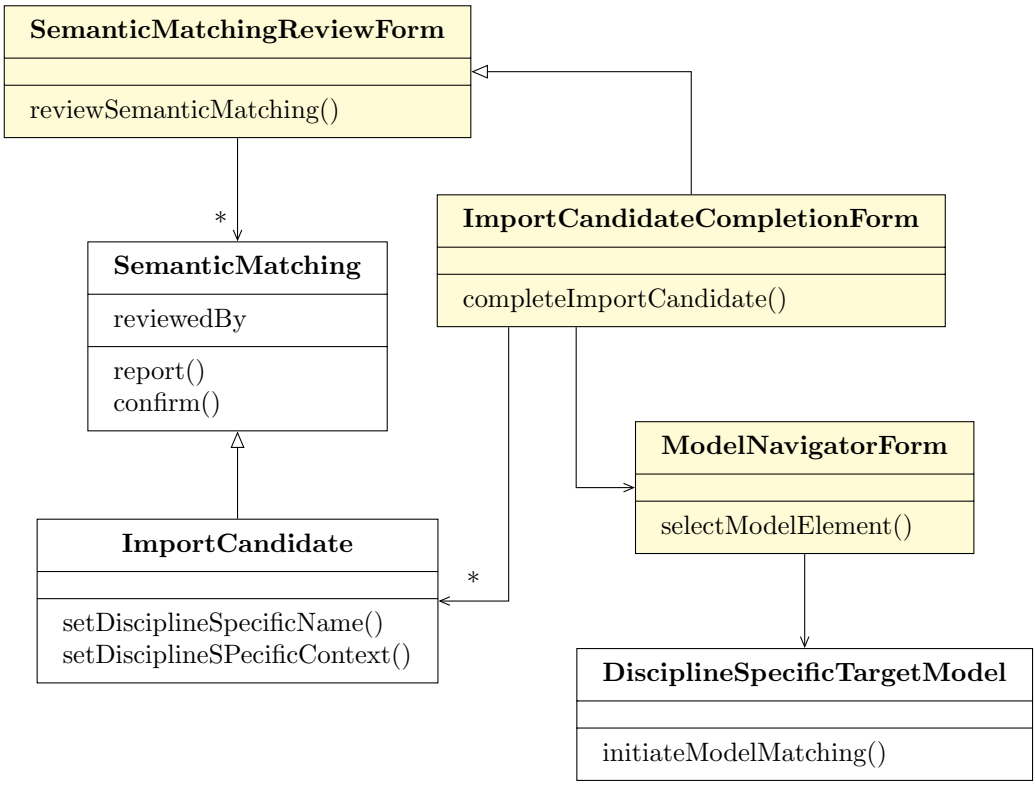

Figure 5.7.: Diagram of Boundary Objects of the Merge Models Partially use case and its relations to other Analysis Objects

Control Object	Definition
`PartialMergingControl`	Manages the flow of events during the Merge Models Partially use case. It triggers the generation and comparison of the `ReferenceConformModels`, generates progress reports, and eventually triggers the propagation of changes.

Table 5.8.: Definition of the Control Object of the Merge Models Partially use case

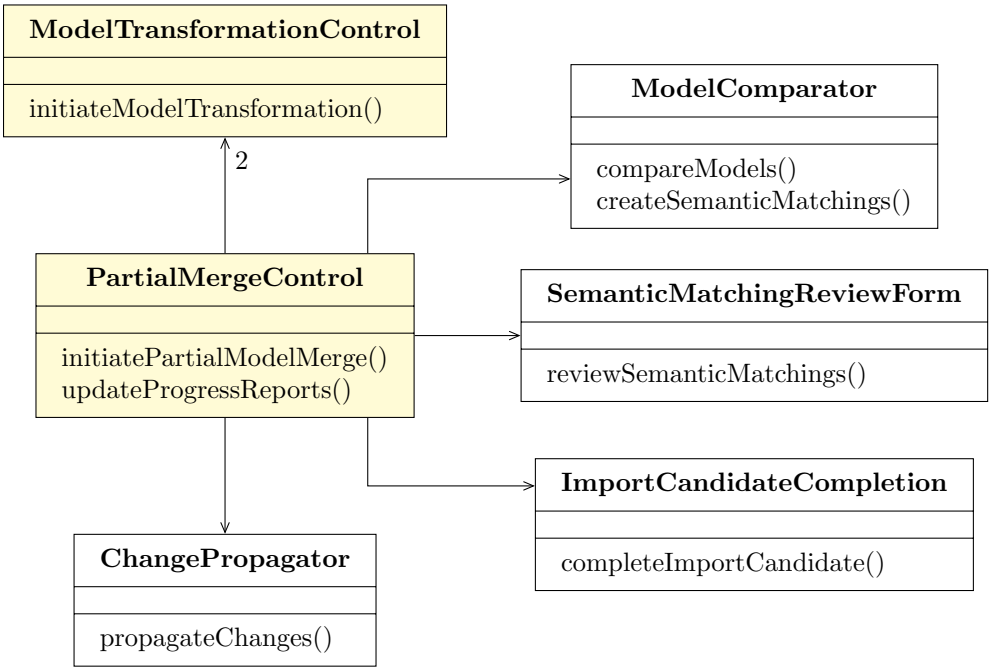

Figure 5.8.: Diagram of Control objects of the Merge Models Partially and its relations to other Analysis Objects

5.4. Analysis Objects of Co-Evolve Integration Artifacts Use Case

The Co-Evolve Integration Artifacts function described in Section 4.3 is similar to the Match Models function. Both functions result in equivalence statements from `DisciplineSpecificConcepts` to the `ReferenceOntology`. Ideally, the Model Owner is only required to match new concepts which have been added to his `DisciplineSpecificModel`, whereas, previously created equivalence statements can be reused. To facilitate this goal the OIDA framework provides components which analyze the components of the framework, artifacts involved in the ontology-based model integration process, and changes known to them, readjusts previous equivalence mappings automatically and support the Model Owner in deciding on ambiguous cases and in creating new equivalence mappings. Especially, the effectiveness of the heuristics employed for the adjustment of existing equivalence relations is crucial for the reuse of work by the Model Owner and, thus, for the overall efficiency of the whole ontology-based model integration. Figure 5.9 depicts the relations of participating Entity Objects of the Co-Evolve Integration Artifacts use case.

The use case is initiated by the `Model Owner` by selecting the `NewDisciplineSpecificModel` he owns and the `OldDerivedOntology` which contains previously made equivalence relations to the `OldReferenceOntology`. This triggers the `NewOntologyGenerator` to generate the `NewDerivedOntology`. The `AdjustmentHeuristic` uses known changes of the OIDA components and artifacts as well as a generic analysis of the artifacts to create adjusted `EquivalenceCandidates`. The `Model Owner` reviews these `EquivalenceCandidates` and creates new `EquivalenceCandidates` manually as described in Section 5.1. Thereby, the `AdjustmentHeuristic` creates recommendations, including previous equivalence relations which could not be adjusted unambiguously. The iterative process of creating and reviewing `EquivalenceCandidates`, the creation of equivalence statements and their validation by the `Reasoner` is carried out as described in Section 5.1. As a result, the `NewDisciplineSpecificModel`, `NewOntologyGenerator`, and `NewReferenceOntology` are consistent, while previously created equivalence relations are reused whenever possible. Table 5.1 lists the Entity Objects involved and a short definition.

The Co-Evolve Integration Artifacts use case is controlled by `CoevolutionControl`. It manages old and new versions of artifacts and triggers the `MatchingControl`. A definition of the Control Objects is given in Table 5.10. `CoevolutionControl` and its relation to other Analysis Objects is depicted in Figure 5.10.

The Boundary Objects of the Match Models use case are reused in this use case.

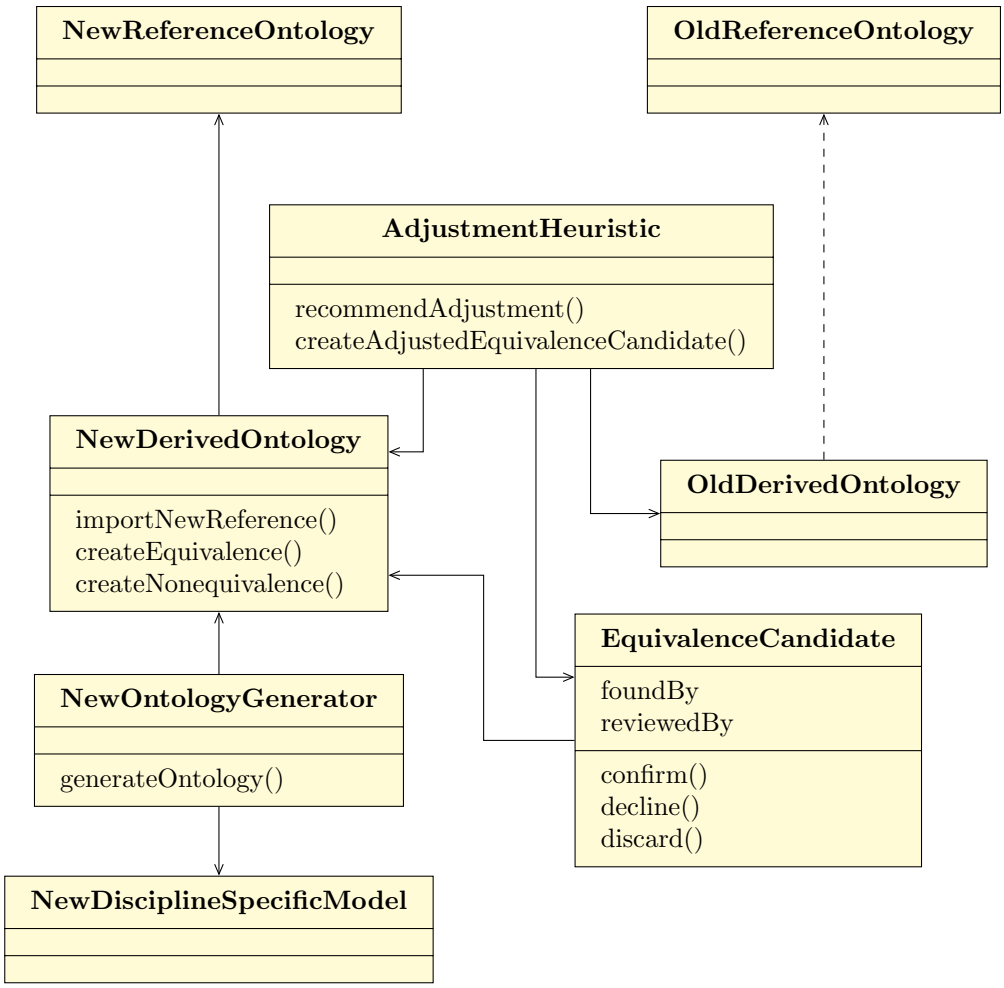

Figure 5.9.: Diagram of Entity Objects of the Co-Evolve Integration Artifacts use case

Entity Object	Definition
NewDisciplineSpecificModel	A new version of a DisciplineSpecificModel described in Table 5.1 modified by or under the supervision of the Model Owner
NewOntologyGenerator	A new version of an OntologyGenerator described in Table 5.1 which has been modified by or is under the supervision of the Ontology Owner
NewDerivedOntology	A kind of DerivedOntology as described in Table 5.1 which has been derived from the NewDisciplineSpecificModel by the NewOntologyGenerator.
NewReferenceOntology	A new version of a ReferenceOntology which is imported by NewDerivedOntology.
OldReferenceOntology	An old version of a ReferenceOntology which was imported by OldDerivedOntology.
OldDerivedOntology	A new version of a DerivedOntology which has been generated by an old version of the OntologyGenerator and from an old version of the DisciplineSpecificModel. It contains equivalence relations to the OldDerivedOntology created by an old Match Models use case.
AdjustmentHeuristic	Creates adjusted EquivalenceCandidates based on existing equivalence and nonequivalence relations in OldDerivedOntology by finding equivalent new concepts automatically using a simple heuristic. Ambiguous findings of the heuristic are passed on to the Model Owner as recommendations for manual adjustment.
EquivalenceCandidate	See Table 5.6

Table 5.9.: Definitions of Entity Objects of the Co-Evolve Integration Artifacts use case

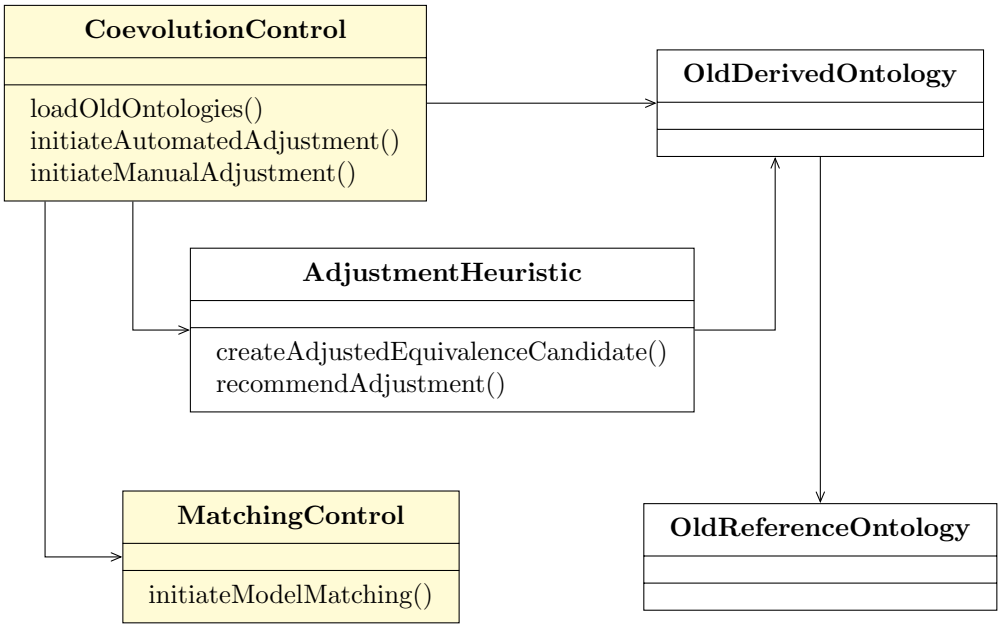

Figure 5.10.: Diagram of Control Objects of the Co-Evolve Integration Artifacts use case and its relations to other Analysis Objects

Control Object	Definition
`CoevolutionControl`	Manages the flow of events during the Co-Evolve Integration Artifacts use case. It manages versions of artifacts, triggers the `MatchingControl`, and generates progress reports.
`MatchingControl`	See Table 5.3

Table 5.10.: Definitions of Control Objects of the Co-Evolve Integration Artifacts use case

6. The Oida Framework Design

This chapter addresses the design goals and the design rationale of the Oida framework with regard to its decomposition. Furthermore, the subsystems of the framework are described in detail.

6.1. Design Goals and Architecture

The design of the Oida framework is based on requirements and constraints stated in Section 3.2 as well as on the following design goals:

User Acceptance Oida has to give the user full control over the ontology-based model integration process especially when assigning formal semantics to model elements. Instead of using complex automated matching algorithms, an easily comprehensible recommendation system must be applied. Furthermore, the framework is designed for a minimal number of user actions. In particular, the user should not have to switch between applications while performing ontology-based model integration. In addition, the user's perception of control should be amplified by a responsive user interface.

Extension of Existing Modeling Frameworks The framework has to be integrated into existing modeling systems. Acknowledging the variety of existing systems in conceptual aircraft design, the framework should not depend on the metamodel of a specific modeling application but on a common meta-metamodel, such as MOF. By using a common meta-metamodel, Oida can reuse general purpose modeling frameworks and still can be integrated as a component in existing conceptual aircraft design tools.

Compliance to Standards Oida must comply to standard formats. This enables file exchange with existing modeling and ontology engineering tools. It allows verification of Oida's transformation algorithms by experts and inspection of models and ontologies in external tools.

The Oida framework is structured in an open layered architecture. The integration system architecture depicted in Figure 6.1 shows eleven subsystems arranged in five layers: `Platform`, `Provider`, `Transformation`, `Service`, and `Application`.

The Platform layer is not part of the Oida framework but represents existing conceptual aircraft design tools and standard modeling frameworks. All Oida subsystems are realized as plug-ins depending on this platform. As a standard ontology framework

is currently not part of any aircraft modeling platform it is integrated into the `OntologyProvider` subsystem in the `Provider` layer. The `Transformation` subsystem in the layer of the same name generates mappings between models, metamodels, and ontologies. The `Services` layer contains subsystems which provide the infrastructure and the services for the ontology-based model integration capabilities. As OIDA integrates the Model Owner as an essential contributor and controller of the OIDA capabilities, the `Application` layer contains subsystems facilitating the use cases Match Models and Merge Models Partially.

6.2. Platform Layer

The `Platform` layer represents existing conceptual aircraft modeling tools. As such, it provides services required by the OIDA framework as shown in Figure 6.1. Accordingly, the assumed components of the `Platform` layer are arranged in an open plug-in architecture. Therefore, the `MOF Service` of the `ModelingFramework` is also available with the `Model Repository` service.

The basis of the `Platform` layer is the `ModelingFramework` which provides the MOF SERVICE which facilitates the dynamic and reflexive creation of models and metamodels consisting of MOF conformal elements. Figure 6.2 shows the model elements which are required by the OIDA framework for model creation and manipulation. Based on the `MOF Service`, the `ModelingClientPlatform` subsystem provides the `MOF Conformal Client Service`, which includes user interface capabilities. Especially, the `ModelNavigatorForm` is reused by the OIDA framework. Thereby, when browsing existing or OIDA-generated models during use cases of ontology-based model, integration the Model Owner operates on a user interface he is familiar with from his experience with the `AircraftModelingApplication`. The `AircraftModelingApplication` adds discipline-specific modeling services. In particular, it provides the `Model Repository Service`. The `FileHandlerStrategy` provides the capability of file-based model exchange with other aircraft conceptual design tools.

6.3. Provider Layer

The `Provider` layer is the connection of an OIDA system to an existing `AircraftModelingApplication` and the host file system. It provides a standardized interface to models, metamodels, and ontologies. In particular, the `OntologyProvider` subsystem connects directly to the host file system whereas the `ModelProvider` connects indirectly via the `AircraftModelingApplication` in the `Platform` layer. It allows file-based data exchange of models and ontologies. Additionally, the `ModelProvider` offers references to existing model elements of the ModelRepository. This allows OIDA subsystems to manipulate existing models in the `AircraftModelingApplication` at runtime.

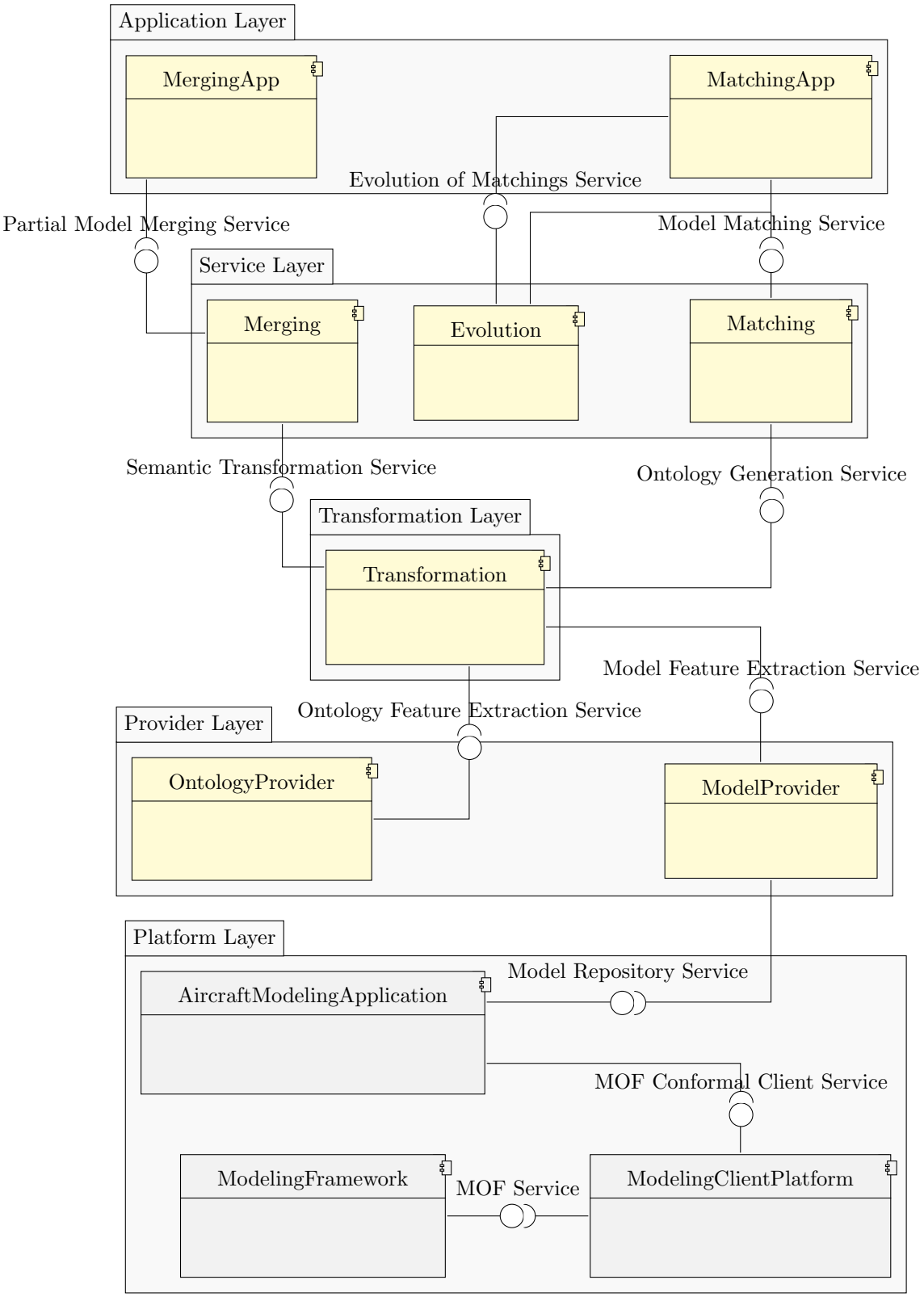

Figure 6.1.: Component diagram of the OIDA framework architecture

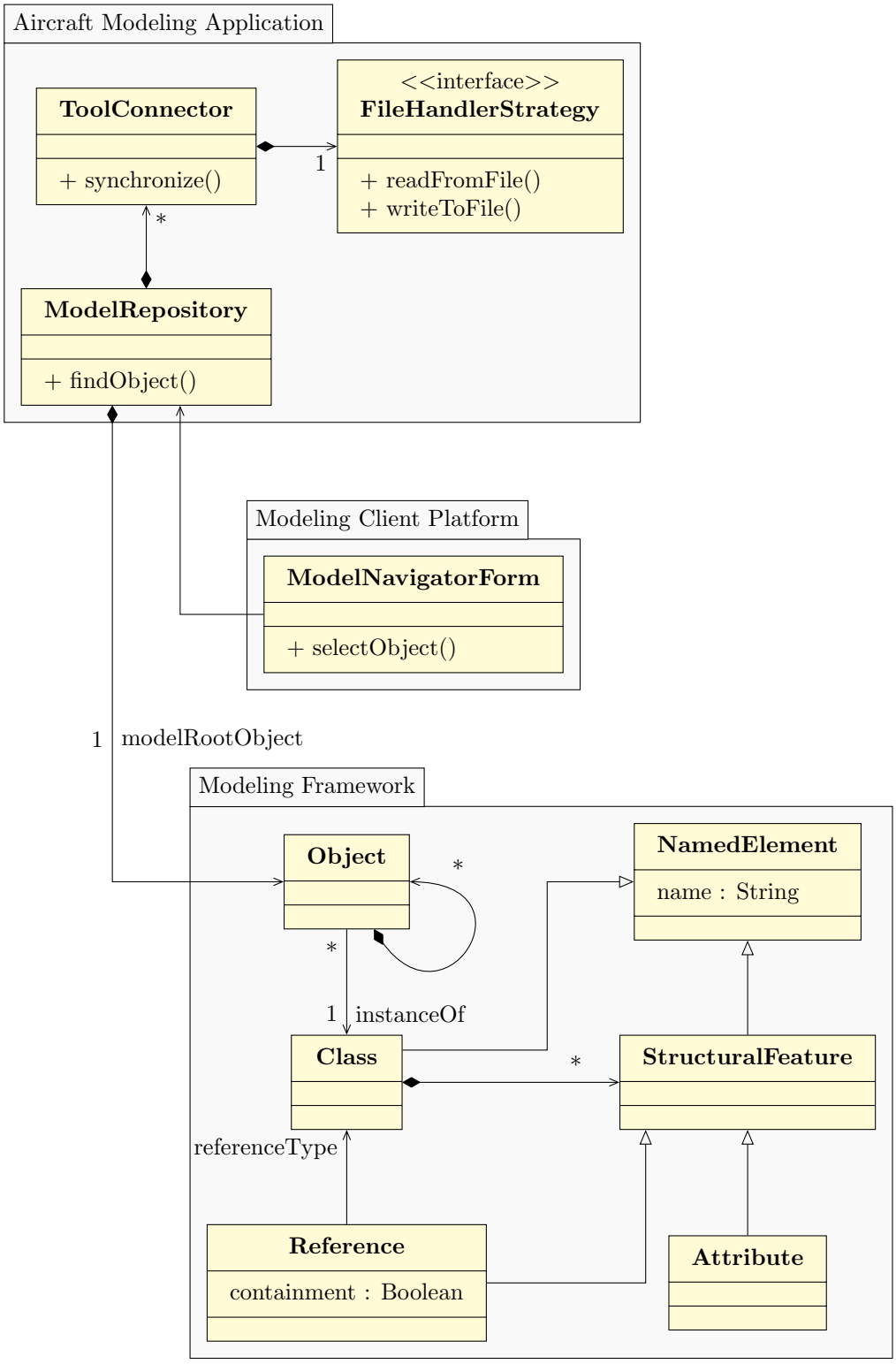

Figure 6.2.: Diagram of important classes of the `Platform` layer

ModelProvider Subsystem

The ModelProvider in the Provider layer contributes to an infrastructure for ModelElementFilters and ModelMetrics. Due to the reflection capabilities of the MOF-based ModelingFramework, these services apply not only to model elements in the ModelRepository on the instance level but also on the metamodel level. A ModelMetrics object (see Figure 6.3) performs measurements on model elements, such as the depth of the containment hierarchy of a certain model. Like ModelElementFilters these need to be resynchronized if the metered model changes. ModelMetrics are not only used in the OIDA framework for progress and performance measurements of Integrate Models use cases, but also for framework maintenance and validation. For instance, a ModelMetric object can measure the number of metamodel classes used by a given disciplineModel. A ModelElementFilter object accesses the ModelRepository and generates references to certain kind of model elements. If the filtered model changes, it must be resynchronized by calling refresh(). For instance, the StructuralFeatures-Filter creates references to all StructuralFeatures, i.e. Attributes and References, used by instanceObjects contained by a specified modelRootObject. Within the OIDA framework, ModelElementFilters are required especially for generating ontologies.

OntologyProvider Subsystem

The OntologyProvider subsystem equips the OIDA framework with an infrastructure for the creation, measurement, and query of ontologies employing an external ontology framework. Figure 6.4 shows that the OntologyProvider is structured similarly to ModelProvider. Following an ADAPTER pattern [Gam+94] OIDA requires an implementation of the interfaces in the OntologyFrameworkAdapter package to an ontology framework. These ADAPTER interfaces are depicted in Figure 6.4 and Figure 6.5. In particular, OIDA requires not only a FACTORY pattern [Gam+94] for OntologyResources and automated reasoning capabilities for validation and inference of generated and modified ontologies. It also requires an OntologyFileHandler for file-based ontology exchange. Similar to the ModelProvider, the OntologyProvider subsystem offers an infrastructure to set up OntologyFilters and OntologyMetrics. OntologyFilters hold references to specific types of ontology resources, such as Individuals, OntologyClasses, or Properties. OntologyMetrics are applied for measuring ontologies for progress reports as well as validation and maintenance of the framework. For instance, an OntologyMetric can measure how many OntologyResources have an equivalence statement to an OntologyResource in the referenceOntology. Both OntologyFilter and OntologyMetric are operated analogously to ModelElementFilter and ModelMetric in the ModelProvider.

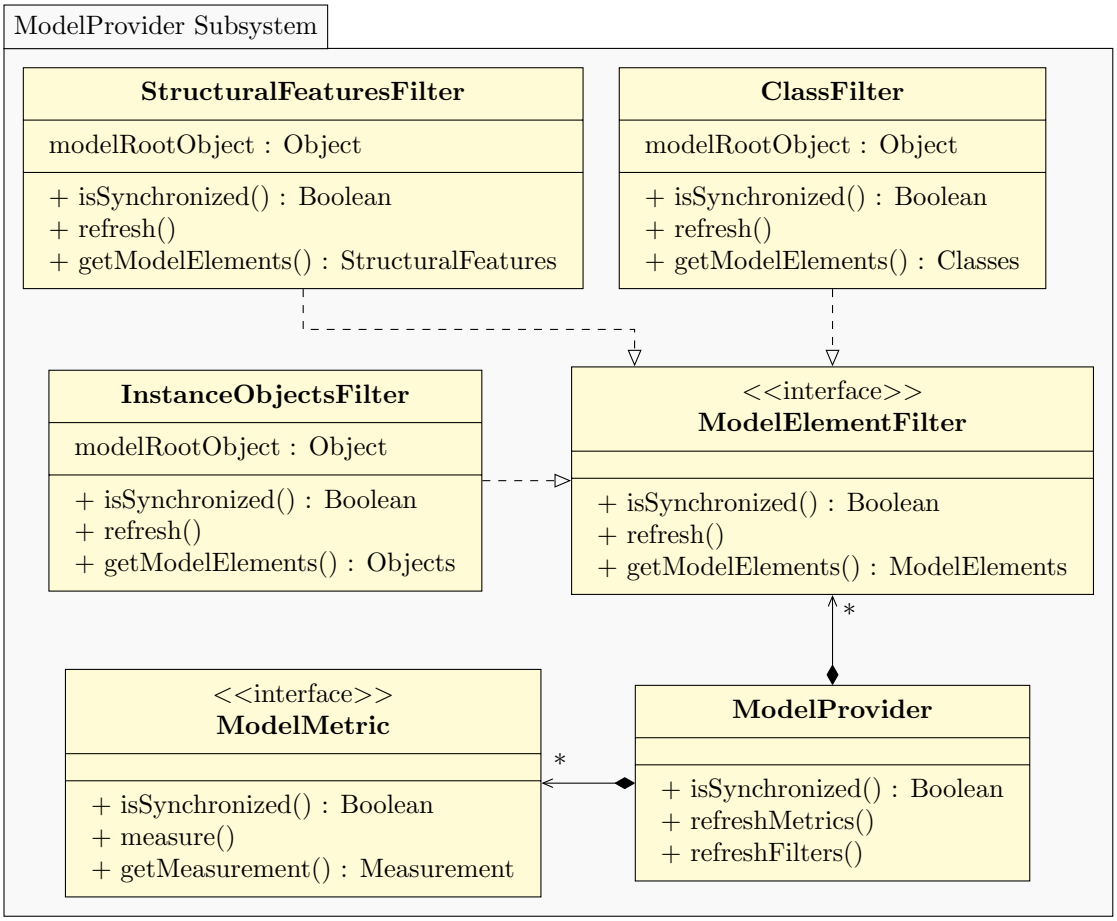

Figure 6.3.: Class Diagram of the `ModelProvider` subsystem

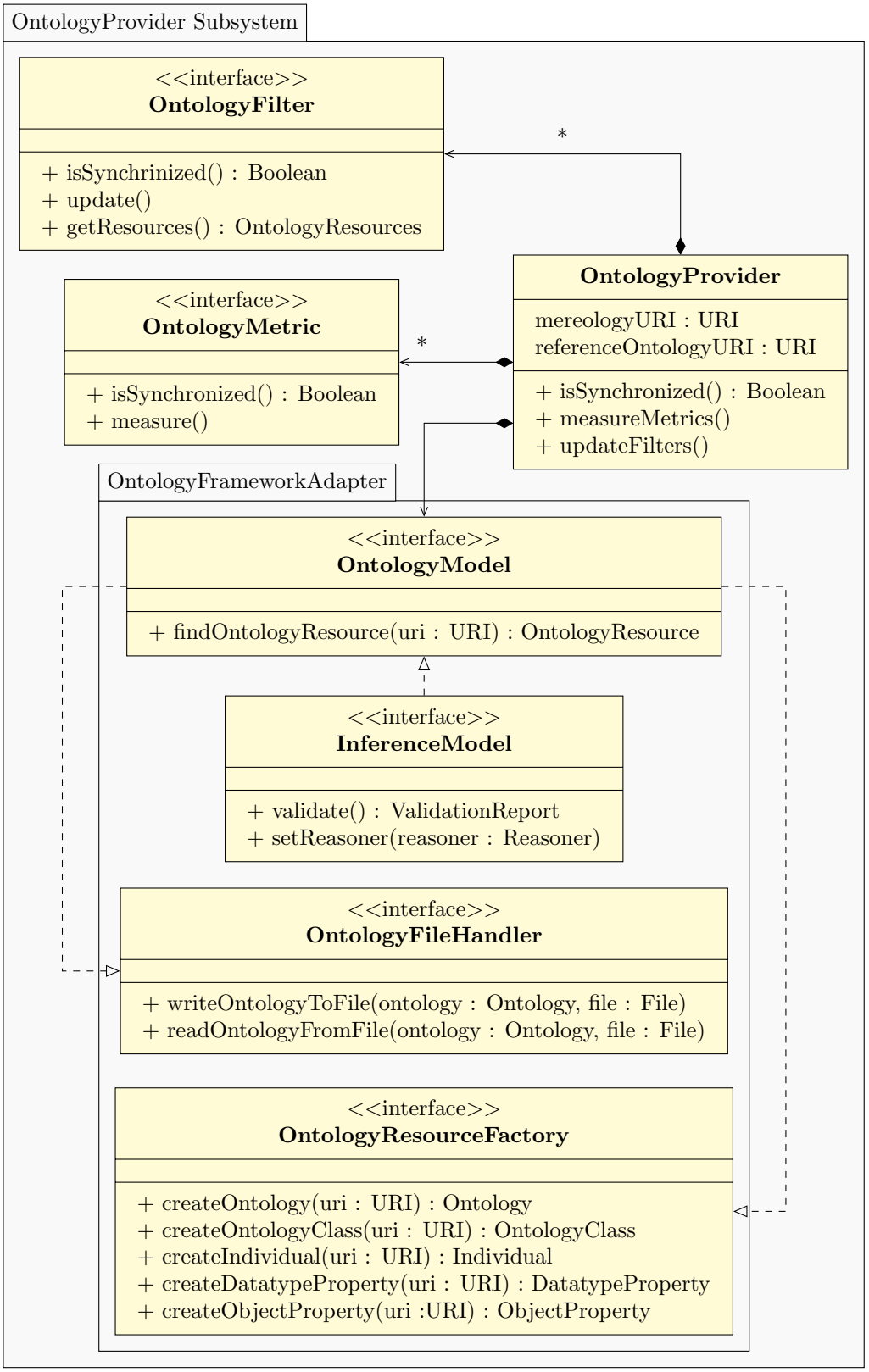

Figure 6.4.: Class diagram of the `OntologyProvider` subsystem

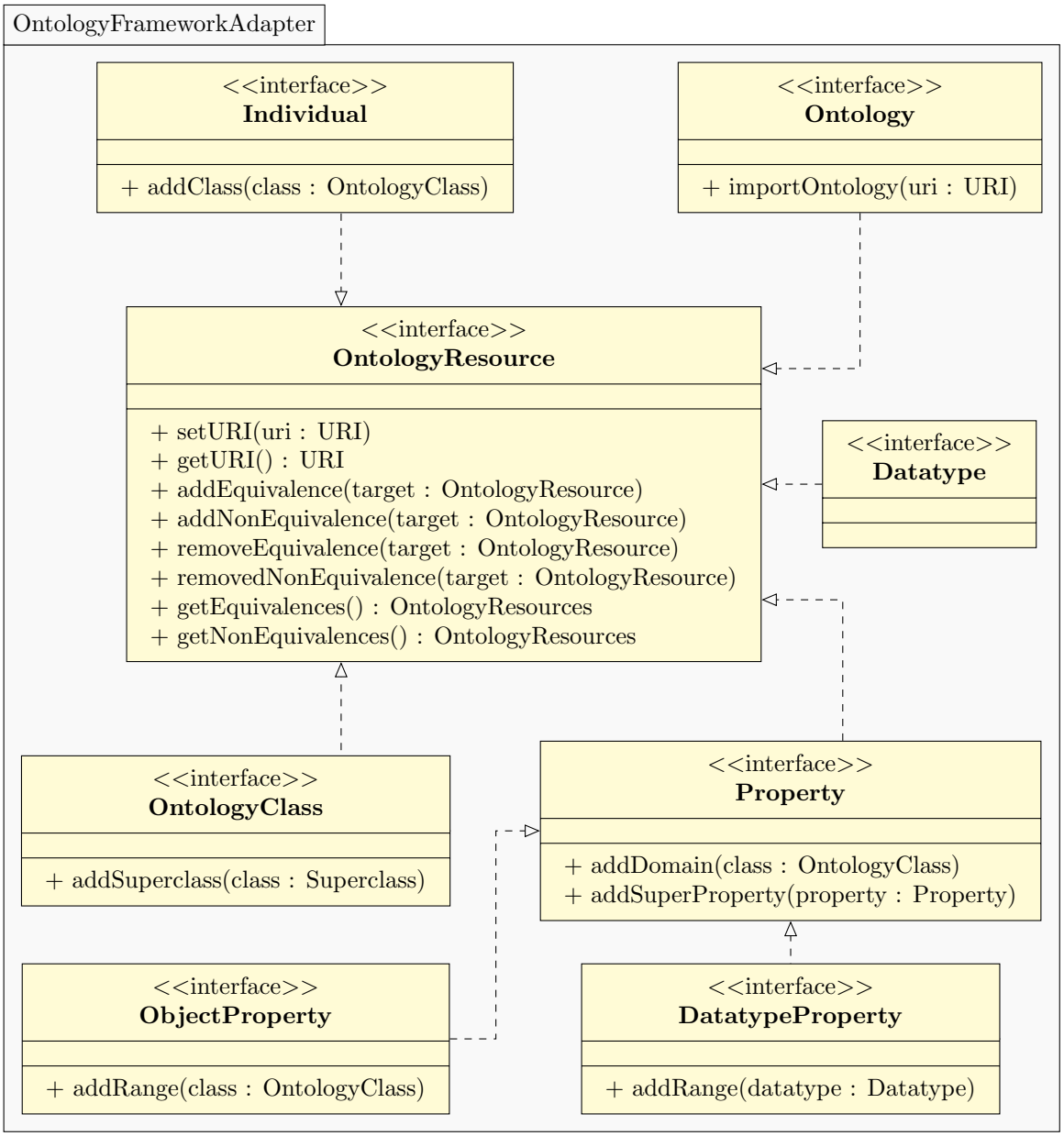

Figure 6.5.: Class diagram of different kinds of `OntologyResources` in the `OntologyFrameworkAdapter` package which is part of the `OntologyProvider` subsystem

6.4. Transformation Layer

The Transformation layer establishes couplings between different disciplineModels and mediates between ontologies and models. This role is realized by the Transformation subsystem.

Transformation Subsystem

The Transformation subsystem provides the Semantic Transformation Service which comprises not only semantic model transformation but also the capability to generate ontologies and metamodels. According to Figure 6.6, TransformationControl holds a set of Transformations which can be interdependent. In order to execute them in the right order, every implementation of a Transformation must determine whether it is executable and whether both the source and the target of the Transformation are synchronized. In general, source and target model are synchronized when the Transformation has been performed and neither its source nor its target have been changed since then. Given that TransformationControl holds instances of ModelTransformation and MetamodelGenerator and startTransformations() is called, TransformationControl executes all Transformations whose isExecuteable() method returns true. MetamodelGenerator will be executed as it does not depend on any other Transformation. ModelTransformation requires the referenceMetamodel which is generated by the MetamodelGenerator as target metamodel. An implementation of isExecuteable() will return the value of isSynchronized() of MetamodelGenerator. When the MetamodelGenerator has finished the generation of the Metamodel and this value changes to true, the TransformationControl executes the ModelTransformation. The TransformationControl references a ModelProvider and an OntologyProvider for the Transformations to operate on.

For instance, the MetamodelGenerator requires the referenceOntology from the OntologyProvider in order to generate the referenceMetamodel in the ModelProvider. Thereby, it registers mappings between ontology and model data types. Conversely, the OntologyGenerator requires a disciplineModel from the ModelProvider and generates a disciplineOntology in the OntologyProvider. During the generation process it employs the Renamer which must generate an unambiguous name from a disciplineObject identifier. This Renamer can be adapted to the particular context of a disciplineModel by a concrete RenamerStrategy. The Renamer can determine if a concrete RenamerStrategy is unambiguous for a given disciplineModel, which allows higher layer clients of the Transformation subsystem to assess the applicability of a particular strategy. Within the Transformation subsystem the Renamer is employed by StatementGenerators for generating ontology statements. For instance, an OntologyClassGenerator extracts all Classes used in the disciplineModel by a ClassFilter of the ModelProvider and generates appropriate OntologyClasses.

Both MetamodelGenerator and OntologyGenerator can operate as unidirectional transformations to facilitate the Integrate Models use cases. The ModelTransformation, however, must operate bidirectionally in order to facilitate the Partial Model

Merge capability which requires not only the transformation of `domainModels` to a common `referenceMetamodel`, but also the propagation of changes in the `referenceConformModels` back to the `domainModels`. Therefore, the `ModelTransformation` creates `ObjectMappings` between `disciplineObjects` and `referenceObjects` at the model level. Furthermore, every object mapping holds a mapping of the respective `disciplineClass` and `referenceClass` and its `StructuralFeatures` provided by EMF. These mappings are entirely based on the equivalence statements in the `disciplineOntology`.

6.5. Service Layer

The `Service` layer contains components which address specific capabilities such as Model Matching, Partial Model Merge, and Co-Evolution. The layer is partitioned into appropriate subsystems, which provide infrastructure and decision support capabilities for higher level subsystems with user interfaces. Whereas the `Matching` and `Evolution` subsystems have the similar purpose of matching artifacts, the `Matching` subsystem focuses on matching concurrent artifacts, whereas `Evolution` matches old and newer versions of ontologies.

Matching Subsystem

The `Matching` subsystem must facilitate the `Semantic Matching` of two `disciplineModels` by the semi-automated matching of a `disciplineOntology` generated from a `disciplineModel` to a `referenceOntology`. The input of this subsystem is an ontology which has been derived from a domain specific model, and a reference ontology which initially can be empty. The `Matching` subsystem provides a framework which creates statements in `derivedOntology` to the `referenceOntology` whereas the `referenceOntology` is extended by new `referenceConcepts` employing the `OntologyProvider`.

The creation of equivalence relationships is challenging especially with respect to the user acceptance. Matching `disciplineOntology` and `referenceOntology` in the given problem context requires a complex algorithm that is hard for Model Owners to follow while the quality of the result cannot be guaranteed. However, there are simple matching algorithms and the Reasoner which can contribute to the manual creation of equivalence relations by recommending and validating partial solutions.

The `Matching` subsystem is designed according to the BLACKBOARD pattern [BHS07]. The `Blackboard` holds a reference to an `OntologyProvider` which contains not only the concepts in the `disciplineOntology` which have been derived from a `disciplineModel`, but also the equivalence statements which are created by the matching subsystem. The `MatchingControl` realizes the `Semantic Matching` service which enables other subsystems to contribute new `KnowledgeSources` and types of intermediate solutions to the `Blackboard`.

Based on the assumption that automated algorithms can only propose `EquivalenceHypotheses` which eventually need to be confirmed by the Model Owner, the `Blackboard` holds intermediate solutions as hypotheses which are generated by a `HypothesisGenerator`. An `EquivalenceHypothesis` references a pair of `OntologyResources` which

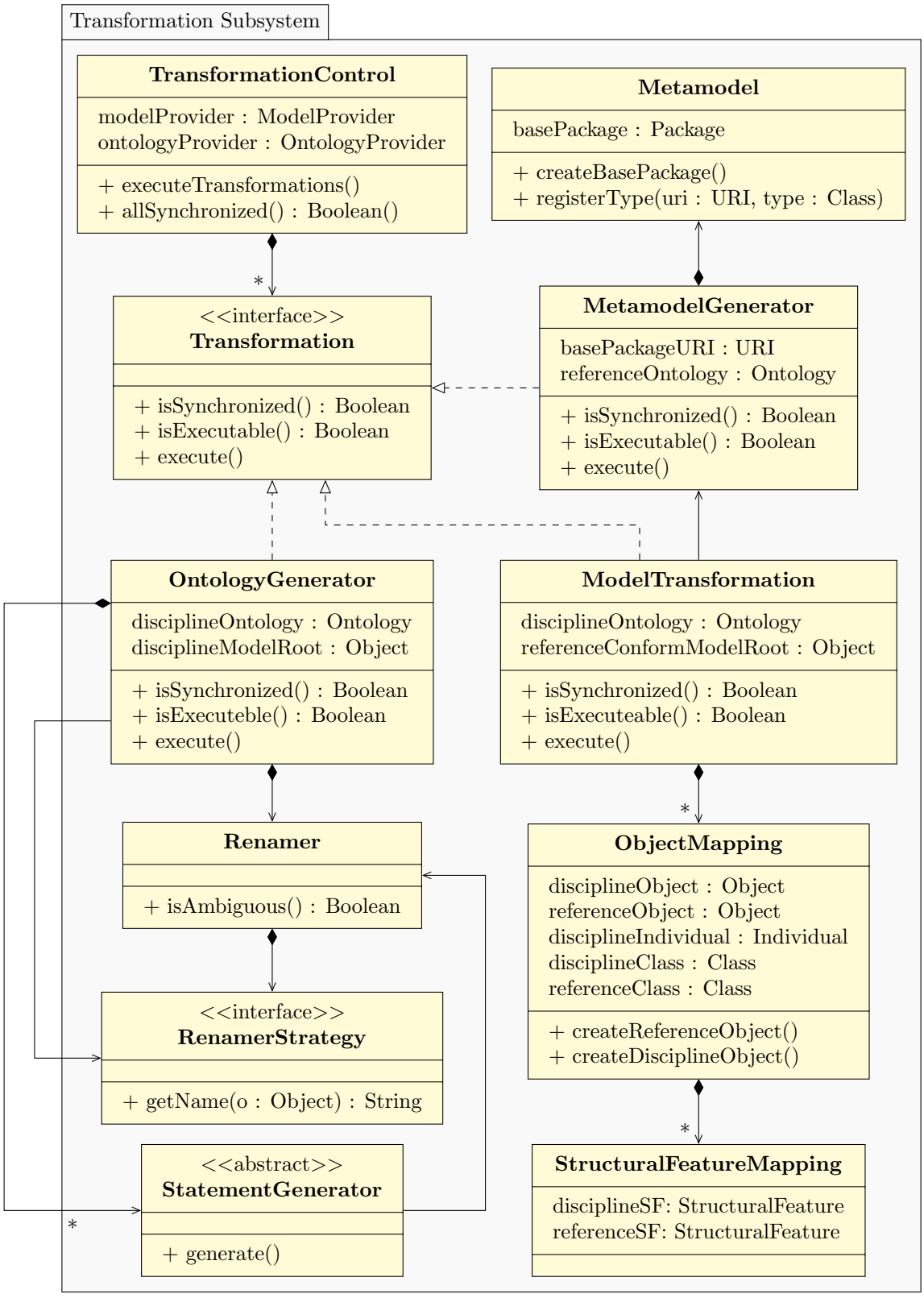

Figure 6.6.: Class diagram of the `Transformation` subsystem

are hypothetically equivalent as well as the URI of the HypothesisGenerator and HypothesisReviewer. HypothesisReviewer and HypothesisValidator are provided with the following review and validation methods: By calling approve() the value of approved is set to true. Conversely, decline() changes the value, approved is changed to false. Both approve() and decline() set the value of reviewed to true which increases the status of an EquivalenceHypothesis from the Proposed Equivalence Hypothesis level to the Reviewed Hypothesis level as shown in Figure 6.8. By calling commit(), the EquivalenceHypothesis is added as a new equivalence or nonequivalence statement to the disciplineOntology depending on the value of the approved attribute. By calling discard() the EquivalenceHypothesis is taken from the blackboard. If an equivalence statement is created, the underlying EquivalenceHypothesis is not deleted from the blackboard. Such an equivalence statement can be removed from the disciplineOntology by calling reconsider(). Generally, an EquivalenceReviewer is operated directly by the Model Owner. As the Matching subsystem does not have a user interface it does not provide concrete EquivalenceReviewers.

Figure 6.8 illustrates the different levels of solutions and the two kinds of KnowledgeSources which contribute to the Blackboard. Automated HypothesisGenerators such as the NameEquivalenceFinder and the StructureEquivalenceFinder automatically contribute to the hypothesis level. In particular, The NameEquivalenceFinder uses the identifiers of the ontology resources as input. It creates EquivalenceHypotheses if two resources are equal. The StructureEquivalenceFinder uses all existing equivalence statements and EquivalenceHypotheses to create new EquivalenceHypotheses if two OntologyResources have equivalent Properties.

The Matching subsystem also provides the SemanticValidator as implementation of the HypothesisValidator interface. As such, the SemanticValidator is an example of an automated HypothesisValidator. It determines whether equivalence statements result in a semantically consistent ontology. A SemanticValidator object uses an InferenceModel provided by the OntologyProvider subsystem. If the Reasoner of this InferenceModel determines inconsistent statements, the SemanticValidator evaluates the validation report of the Reasoner. If a particular equivalence statement causes an inconsistency, the SemanticValidator removes the equivalence statements from the ontology and puts the equivalence relation as a proposed and unapproved EquivalenceHypothesis on the Blackboard for reconsideration. If the Reasoner determines that a particular equivalence statement is consistent with the existing ontology it removes the respective equivalence statement by calling the discard() method.

Merging Subsystem

The PartialModelMerge service of the Merging subsystem is provided and initiated by MergingControl. For the Merge Models Partially use case MergingControl requires a reference to the source and target disciplineModels which both need to be transformed to a common MetaModel. Therefore, the MergingControl requests the Semantic Model Transformation service from the Transformation subsystem. This service is configured

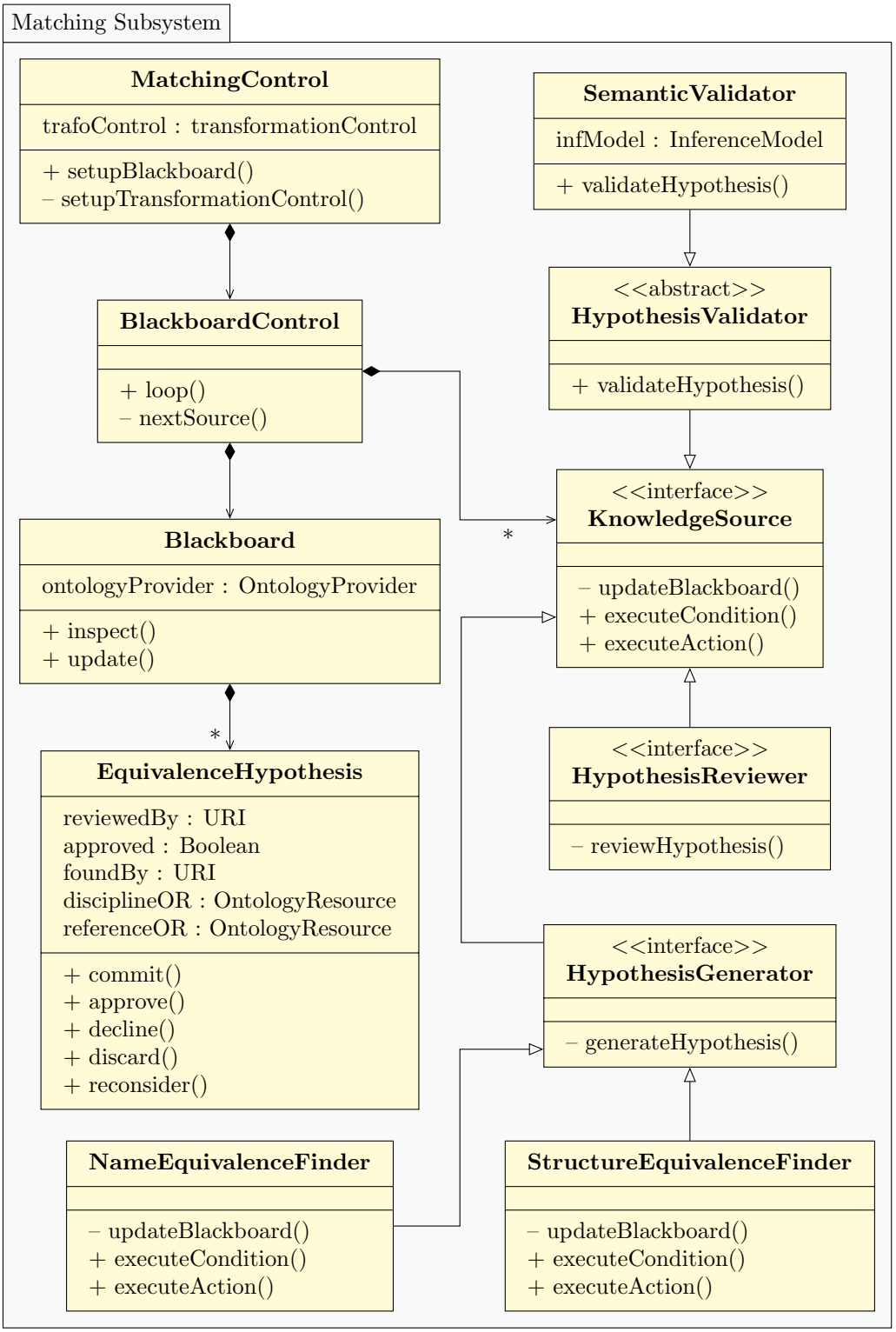

Figure 6.7.: Class diagram of the `Matching` subsystem

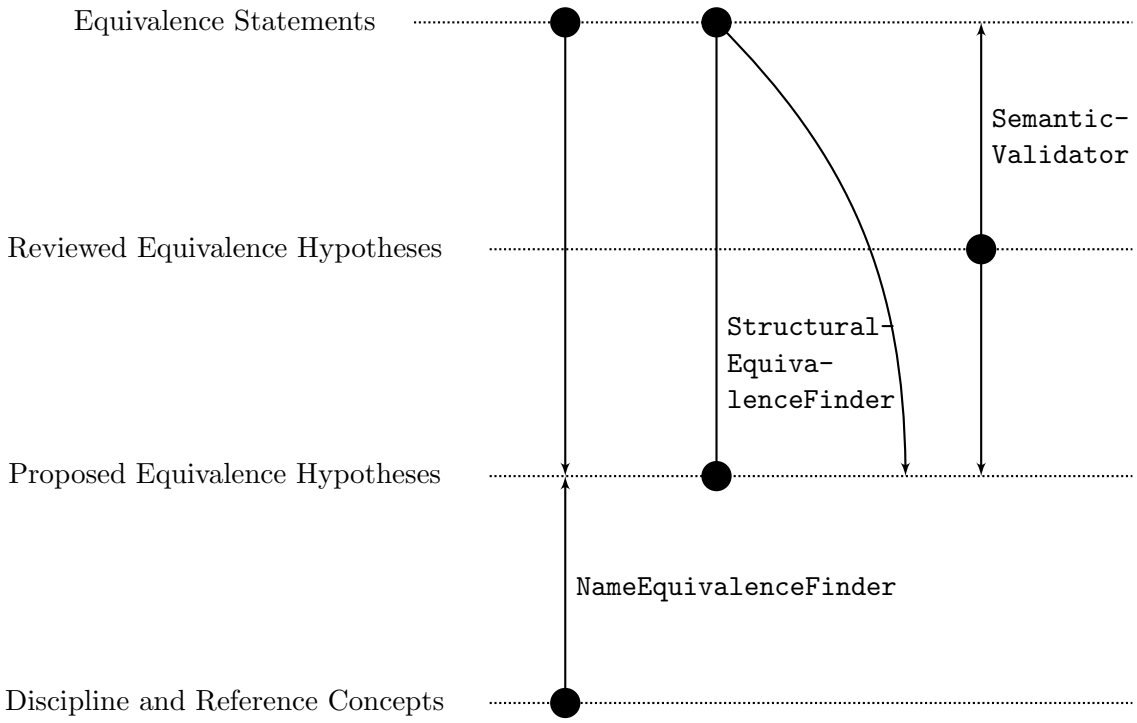

Figure 6.8.: Different levels of inputs and outputs of the `KnowledgeSources` provided by the `Matching` subsystem. `NameEquivalenceFinder` and `Structural-EquivalenceFinder` combine input from two levels to generate Proposed Equivalence Hypotheses. The `SemanticValidator` also creates Equivalence Statements.

by creating a MetamodelGenerator and two ModelTransformations for the source and the target disciplineModel, respectively.

The MergingControl sets the ontologyProvider reference of the TransformationControl to the OntologyProvider which contains the DerivedOntologies of target and source model. The OntologyResources of the DerivedOntologies are mapped to the ReferenceOntology by equivalence statements. The disciplineOntology attribute of the targetModelTransformation and sourceModelTransformation are set to the URI of the respective derivedOntologies. The MergingControl calls the startTransformations() method. As a result the targetModelTransformation and sourceModelTransformation have a reference to the root object of the respective model conformal to the referenceMetamodel and ObjectMappings between discipline-specific objects and reference-conformal objects.

When all Transformations are synchronized, the MergingControl creates a ModelComparator and sets an ObjectComparator implementation as a strategy to determine whether two objects are semantically equivalent. Then MergingControl calls the compareModels() method using the rootObjects of the reference conformal source and target models as parameters. The ModelComparator uses the equivalence statements in the DerivedOntologies to determine semantically matching objects. If it finds two semantically matching objects, the ObjectComparator also determines whether these objects have conflicting states. If the two objects do not have conflicting states, a SemanticMatch is created which has a reference to the ObjectMapping of the source and the target model. If the object states are in conflict, a ConflictCandidate is created. If an object from the source model cannot be correlated to an equivalent target object, an ImportCandidate is created, which initially only references the sourceObjectMapping.

After this initialization the ModelComparator holds a list of SemanticMatchings between the source and the target disciplineModels which can be reviewed by the Model Owner. The user interface for review and decisions on ImportCandidates and ConflictCandidates is not facilitated by the Merging subsystem but by subsystems on higher layers of the OIDA framework. In particular, if the resolveConflict() method is called, the state of the referenceConformTargetObject is replaced by the state of the referenceConformSourceObject. The respective ObjectMapping is used to propagate the new state of the referenceConformTargetObject to the DisciplineSpecificSourceObject. If the disciplineName and disciplineContainer are set and the importCandidate() is called, the targetModelTransformation creates a new ObjectMapping taking a copy of referenceConformSourceObject as referenceConformTargetObject. The targetModelTransformation creates the appropriate disciplineSpecificTargetObject using the disciplineName and disciplineContainer.

Evolution Subsystem

The Evolution subsystem supports the Co-Evolve Integration Artifacts use case by additional KnowledgeSources. The inputs of these KnowledgeSources are coupled artifacts which have become inconsistent due to modifications which have not been propagated to all other artifacts. In an OIDA-facilitated model integration process most of the artifacts

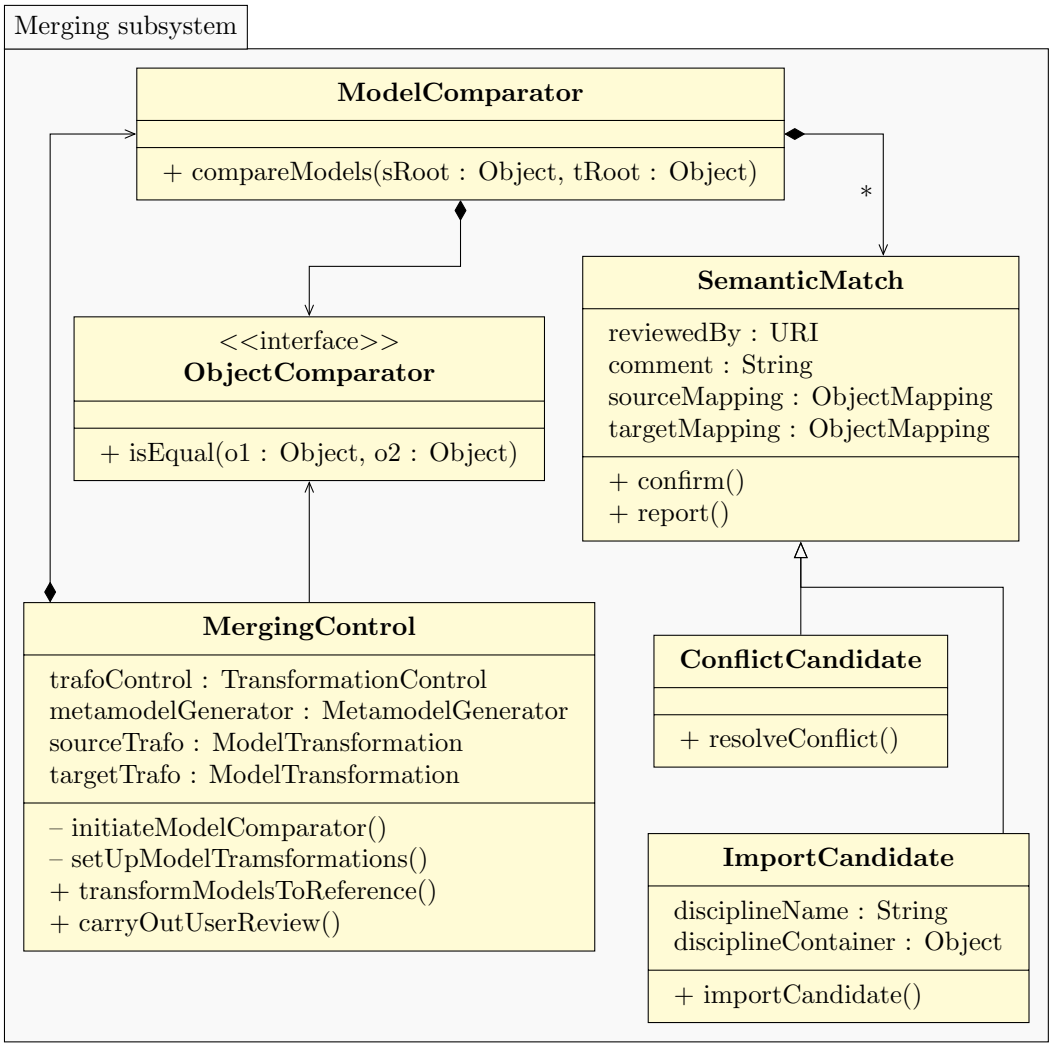

Figure 6.9.: Class diagram of the `Merging` Subsystem

are generated. Only the equivalence statements need to be created interactively. Therefore, the KnowledgeSources of the Evolution subsystem are focused on the migration of previously approved equivalence statements.

The Evolution subsystem supports two scenarios. In the first scenario the modifications on the artifacts are recorded. An algorithm evaluates the modifications and performs appropriate modifications on the coupled artifacts to re-establish consistency. In the second scenario the modifications which led to an inconsistency are unknown. In this case, the inconsistencies must be detected and resolved by a heuristic which does not guarantee results, or that any results will be unambiguous.

In both cases the previous version of the disciplineOntology and the referenceOntology can be loaded into the OntologyModel of the OntologyProvider held by the Blackboard. Furthermore, it is assumed that available modifications on artifacts are expressed as statements in the OntologyModel.

As illustrated in Figure 6.10, the Evolution subsystem provides an EvolutionControl and the two KnowledgeResources: EvolutionAlgorithm and EvolutionHeuristic. EvolutionControl manages the availability of previous equivalence statements and known modifications as basic input of the Blackboard. Figure 6.11 illustrates the KnowledgeSources implementing the EvolutionAlgorithm and the EvolutionHeuristic interface. The EvolutionAlgorithm evaluates known modifications and revises equivalence statements accordingly. The algorithmic character of this KnowledgeSource implies its results are unambiguous. Therefore, an EvolutionAlgorithm generates automatically approved EquivalenceHypotheses. An EvolutionHeuristic, however, evaluates both existing disciplineConcepts and referenceConcepts as well as previous equivalenceMappings and applies a heuristic in order to adjust them to the new version. Depending on the result of the heuristic an implementation of the EvolutionHeuristic can produce approved or unapproved EquivalenceHypotheses.

6.6. Application Layer

The Application Layer provides user interfaces for the Model Matching and Partial Model Merge capabilities. Like the other components of the OIDA framework, the user interface components are integrated into the modeling environment and exhibit a familiar look-and-feel for the aircraft modeler in order to ensure that the user experiences the use cases of ontology-based model integration as extensions of his aircraft design environment rather than as a separate application.

MatchingApp Application

The MatchingApp application enables the Model Owner to match a given disciplineModel to a referenceOntology by declaring equivalence relationships between discipline and reference concepts employing the services of the Matching subsystem. Figure 6.13 illustrates how the MatchingApp initializes the Blackboard in the Matching subsystem setting up the TransformationControl and the KnowledgeSources. The

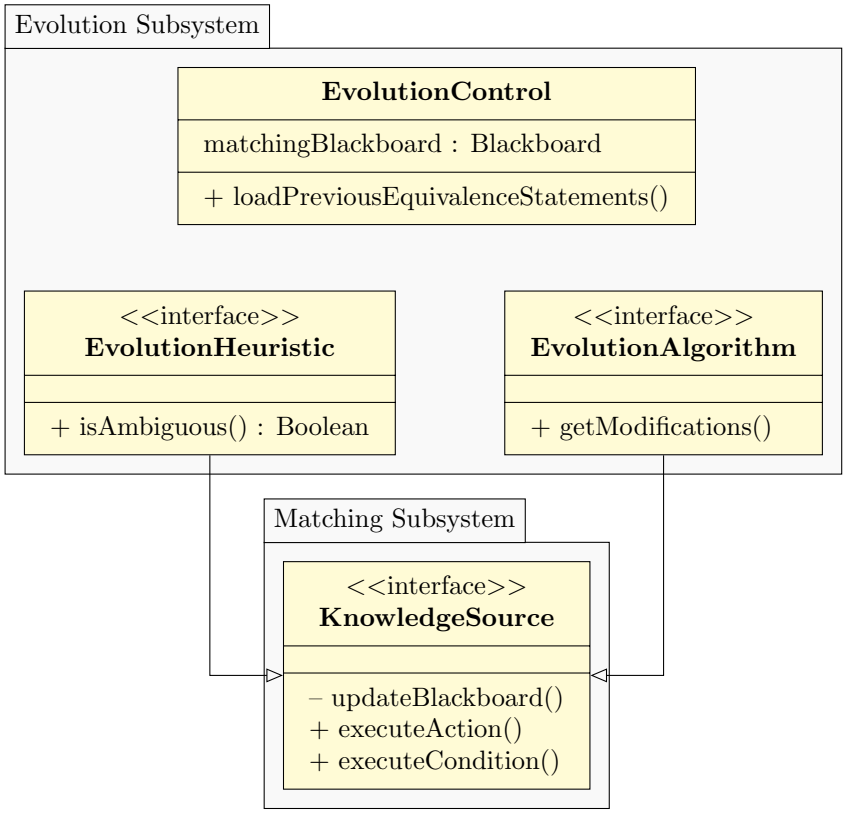

Figure 6.10.: Class diagram of the `Evolution` subsystem

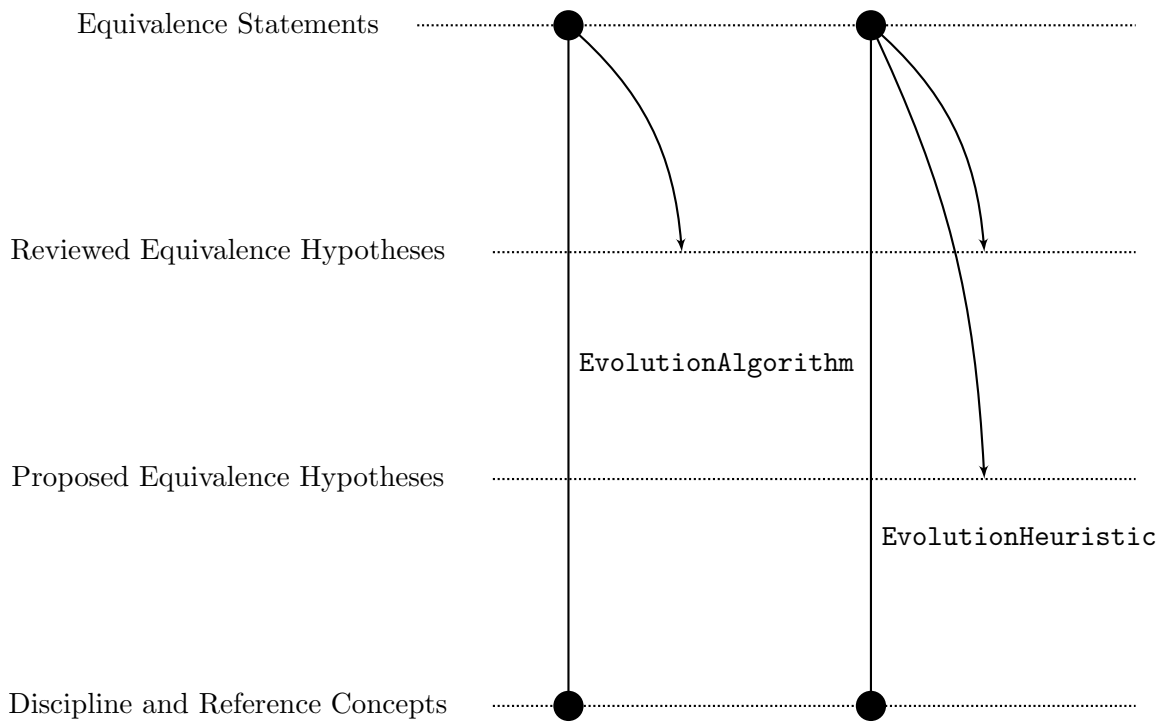

Figure 6.11.: Different levels of inputs and outputs of `KnowledgeSources` provided by the `Evolution` subsystem. Both `EvolutionAlgorithm` and `EvolutionHeuristic` process Discipline and Reference Concepts as well as existing Equivalence Statements and create Reviewed `EquivalenceHypotheses`. `EvolutionHeuristic` also proposes `EquivalenceHypotheses` for reconsideration.

TransformationControl is required to generate the input for the Matching subsystem, i.e. disciplineOntology derived from a given disciplineModel. Additionally, it adds interactive KnowledgeSources such as the ManualEquivalenceGenerator and the EquivalenceReviewForm. The ManualEquivalenceValidator provides a user interface which displays EquivalenceHypotheses on the Blackboard and can receive and carry out the manual review of these EquivalenceHypotheses by the Model Owner.

The MatchingApp also provides the BlackboardControlForm for the control of the Blackboard. It allows the user to adjust the behavior of the nextSource() method by the manual activation and deactivation of particular KnowledgeSources. Furthermore, it also provides a report on the progress of the Match Models process. After the initialization, MatchingApp triggers the generation of the disciplineOntology and calls the loop() method of the BlackboardControl to start the interactive semi-automated matching process. It also saves the results when the user determines that the disciplineOntology is sufficiently well matched with the referenceOntology.

Figure 6.12 illustrates how these KnowledgeSources contribute to the different solution levels held by the Blackboard. The ManualEquivalenceGenerator provides a user interface which allows the Model Owner to navigate the derivedOntology and the referenceOntology and select pairs of OntologyResources in order to create EquivalenceHypotheses manually. If the Model Owner cannot find an adequate reference concept, the ManualEquivalenceGenerator provides a ReferenceCreatorDialog which the Model Owner can use to add new referenceResources to the referenceOntology.

The ManualEquivalenceReviewer provides a user interface displaying the available EquivalenceHypotheses to the Model Owner. The Model Owner reviews the hypotheses by either approving, declining, or discarding them. As a user interface equipped KnowledgeSource, the ManualEquivalenceReviewer contributes with these user decisions to the Blackboard of the Matching subsystem.

MergingApp Application

The MergingApp application enables the Model Owner to review the SemanticMatches, ConflictCandidates, and ImportCandidates. Figure 6.14 shows the classes provided by the MergingApp subsystem. The MergingApp uses the SemanticMatchReview both for reviewing SemanticMatchings and ConflictCandidates. The ImportCandidate-CompletionForm is a specialization of the SemanticMatchReview which allows the Model Owner to complete the information required for importing the object by choosing a discipline-specific name and container object for the ImportCandidate object. For the selection of the container, the ContainerNavigatorForm, a specialized ModelNavigatorForm of the ModelingClientPlatform is used that only displays container objects of the targetDisciplineModel. During the Merge Models Partially use case the user interface displays a report on the progress of the review. As the ModelProvider subsystem works directly on the targetDisciplineModel the conflict resolution and import decisions are carried out instantly. However, when the user closes the MergingApp, review comments and newly imported equivalence statements are stored in the target-DisciplineOntology.

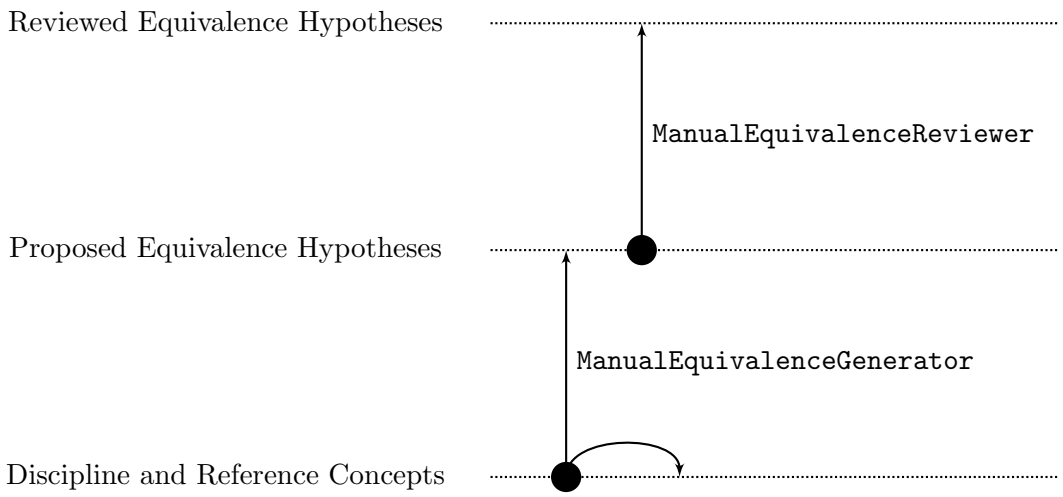

Figure 6.12.: Different levels of inputs and outputs of automated and interactive `KnowledgeSources` employed by the subsystem. Unlike the `Matching` subsystem the `KnowledgeSources` of the `MatchingApp` do not create equivalence statements.

6.7. Oida Knowledge Sources: Overview

The `Matching` of the OIDA framework is designed to facilitate the matching of a `disciplineOntology` derived from a `disciplineModel` to a `referenceOntology` by using a BLACKBOARD pattern. The BLACKBOARD architecture allows not only the `Matching` subsystem but also other subsystems like `Evolution`, `MatchingApp` to contribute to the semi-automated Match Models use case. Figure 6.15 gives an overview on `KnowledgeSources` described in this chapter.

In order to give the Model Owner control over the Match Models process, `EquivalenceHypotheses` can only be approved by interactive components. An exception is the `EvolutionAlgorithm` which automatically adjusts previously created equivalence statements based on known modifications, such as Rename refactorings.

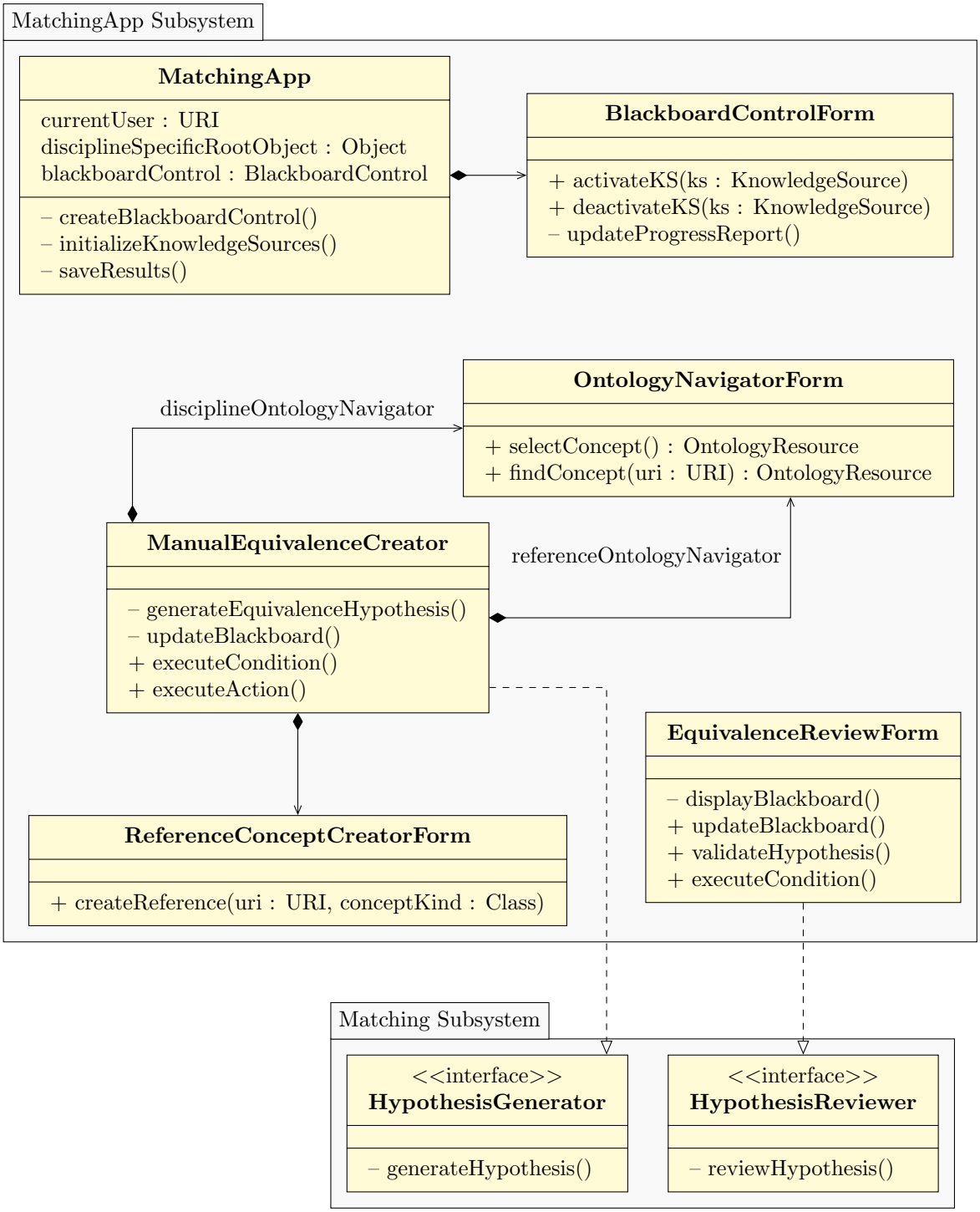

Figure 6.13.: Class diagram of the `MatchingApp` appliaction

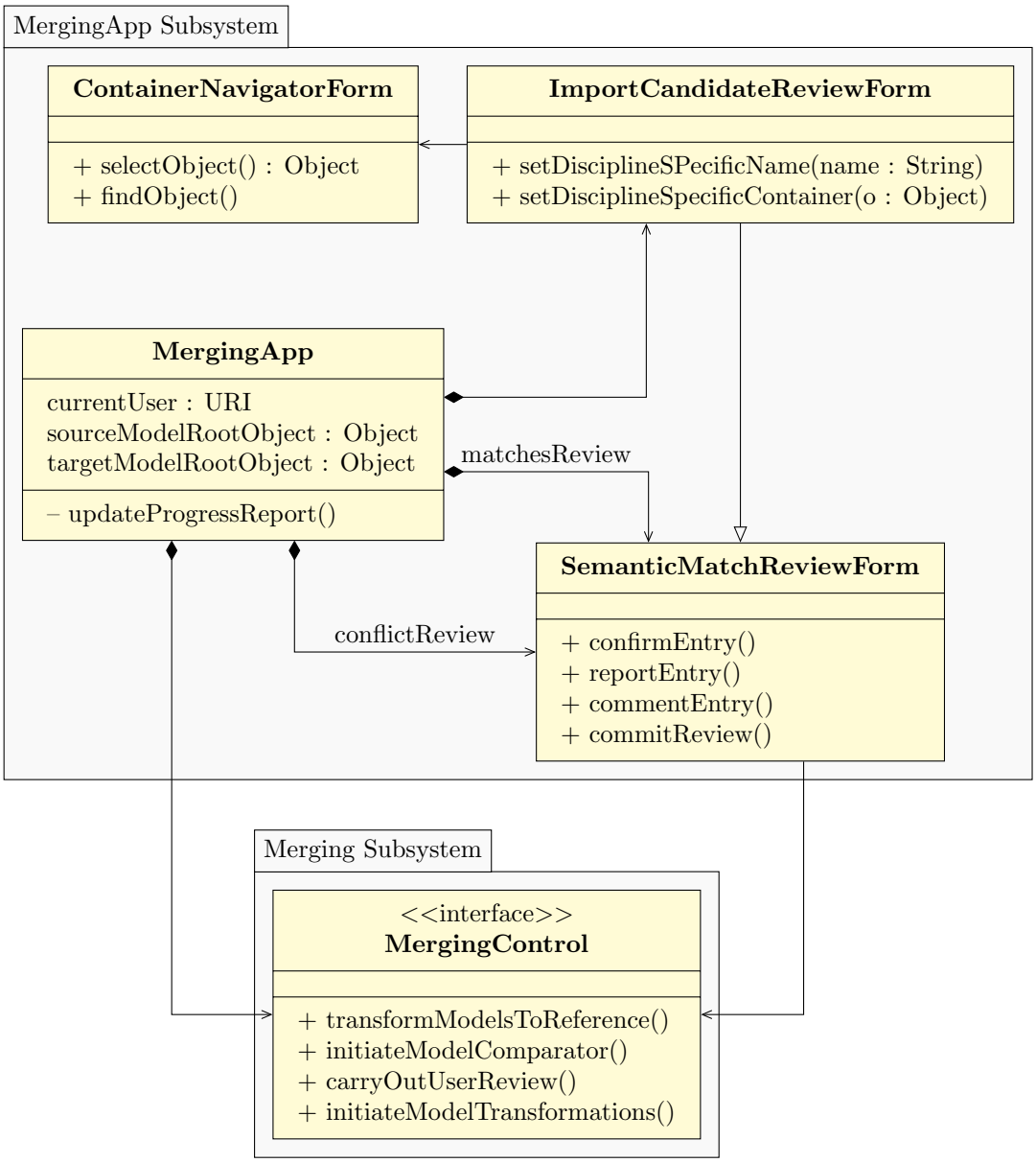

Figure 6.14.: Class diagram of the `MergingApp` application

KnowledgeSource	Description
	Matching subsystem
NameEquivalenceFinder	Creates unapproved EquivalenceHypotheses based on equal names of discipline and reference concepts considering existing equivalence statements.
StructuralEquivalenceFinder	Creates unapproved EquivalenceHypotheses based on existing equivalence statements on structural features.
SemanticValidator	Validates approved EquivalenceHypotheses by using an InferenceModel of the disciplineOntology. Depending on the Validation Report of the Reasoner it creates equivalence statements or re-proposes EquivalenceHypotheses for reconsideration.
	Evolution subsystem
EvolutionAlgorithm	Redirects previously stated equivalences from newly generated disciplineOntology to new referenceOntology based on known modifications on integration relevant artifacts by creating approved EquivalenceHypotheses.
EquivalenceHypothesis	Redirects previously stated equivalences from a newly generated disciplineOntology to new referenceOntology based on a heuristic by creating approved or unapproved EquivalenceHypotheses, depending on the ambiguity of the heuristic.
	MatchingApp subsystem
ManualEquivalenceGenerator	Provides a user interface which allows the Model Owner to create unapproved EquivalenceHypotheses.
ManualEquivalenceReviewer	Provides a user interface which allows the Model Owner to review EquivalenceHypotheses.

Table 6.1.: Definitions of OIDA KnowledgeSources

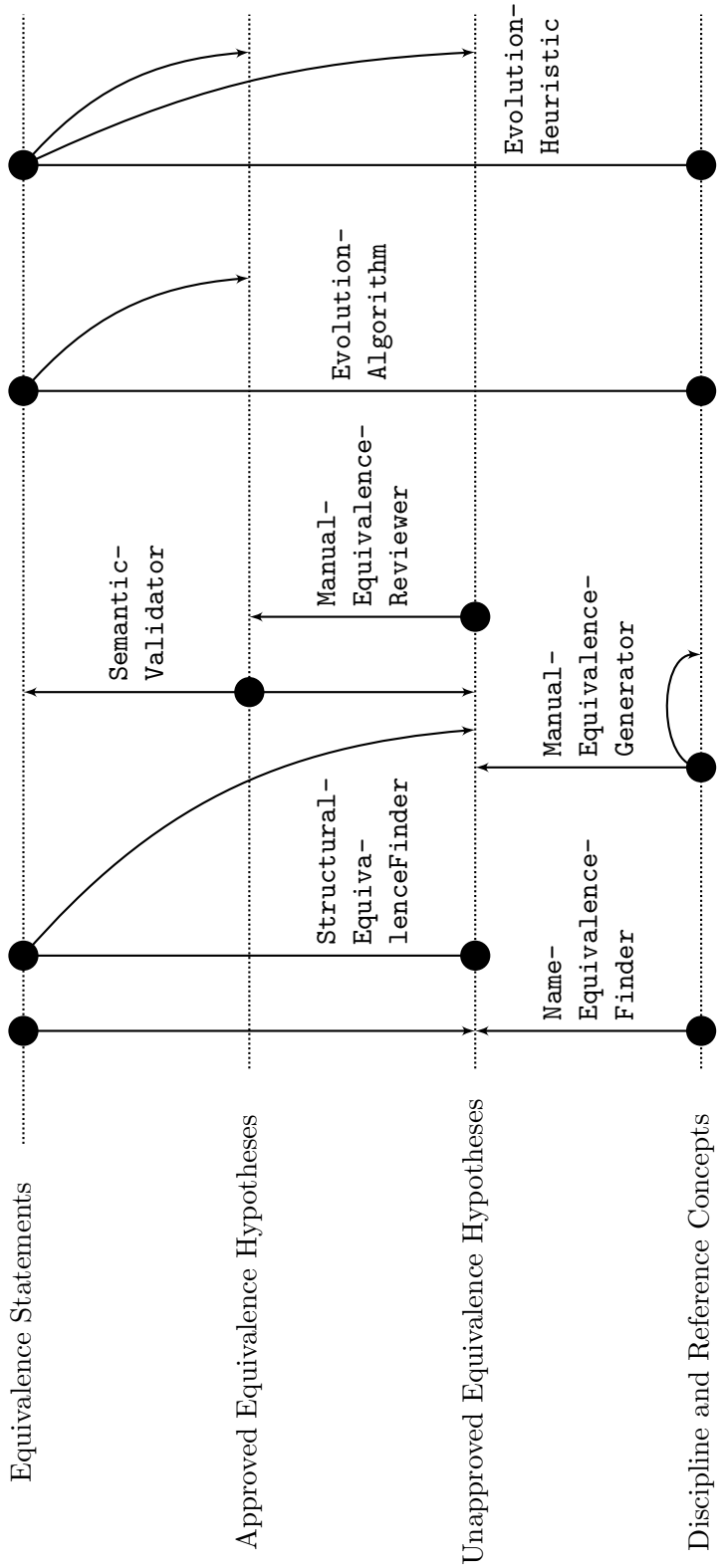

Figure 6.15.: Depiction of solution layers and `KnowledgeSources` provided by the `Matching` and the `Evolution` subsystems as well as by the `MatchingApp` application

7. Evaluation

The objective of this evaluation is to analyze the OIDA framework in typical scenarios of ontology-based model integration regarding effectiveness, efficiency, and plausibility of results from the perspective of typical users. In particular, the OIDA framework is evaluated with regard to to the following criteria which are derived from the requirements stated in Section 3.2:

- As the solution transfers the problem of model integration from the modeling technological space to the ontology technological space, the mapping from model to ontology must be injective in order to guarantee that no relevant information is lost in the transformation. In particular, the unambiguity of the `RenamerStrategy` of the transformation T_{MO} must be validated.
- The OIDA framework must demonstrate its capability to identify conflicts between the discipline-specific sample models APD, APA, and SIMCAD. Furthermore, the OIDA framework must demonstrate its capability to resolve conflicts using the SIMCAD sample model as Chief Engineering Model which overrides the other sample models.
- The Model Owners must perceive the OIDA-facilitated interactive process of ontology-based model integration as efficient. From their point of view the results of the integration process must be plausible.
- The OIDA framework must demonstrate its co-evolution capabilities, i.e. small modifications in artifacts such as source models, the reference ontology, or the OIDA implementation, which have become coupled by performing ontology-based model integration, must require only small effort by the Model Owners to reestablish consistency between these artifacts.

7.1. Evaluation Design

The evaluation is structured into four parts. In the first part, the effectiveness of the model-to-ontology transformation as a critical capability of the OIDA framework is evaluated. The next three parts evaluate the Model Matching, Partial Model Merge, and Co-Evolution capabilities of the OIDA framework.

The OIDA framework explicitly addresses the domain context of aircraft conceptual design and the importance of efficiency and plausibility of results as perceived by the Model Owner. Therefore, the evaluation requires empirical evaluation strategies appropriate for this domain context. Wohlin et al. [Woh+00] describe survey, case study, and experiment as basic empirical strategies:

A *survey* draws a representative sample from a population. Quantitative and qualitative data are collected by questionnaires and interviews. The data are analyzed in order to generalize the conclusion to the entire population. In empirical software engineering, surveys are typically used to gather data about the state, the opinion, or performance of a population before or after a new software engineering method or tool is introduced. In contrast to a case study, a survey puts more emphasis on the population than on the specific sample.

In a *case study*, a typical situation or project is analyzed in which a solution or method of interest is used. In a case study, measurements and analysis emphasize the peculiarities of the particular case. However, the conclusions from case studies cannot be generalized. In most case studies, comparative analysis is the most important method to strengthen the validity of conclusions. In particular, comparative case studies are important to filter co-founding factors. For instance, the benefit of a new tool can be evaluated by a case study in which two groups have to perform the same task. The first group uses the new tool exclusively while the second group uses a conventional tool. If the performance of the first group is considerably better, the new tool is most likely the relevant factor for the success.

The objective of an *experiment* is to achieve statistically significant conclusions by systematic and quantitative measurements and by treatments employing techniques like randomization, balancing, and blocking. Randomization means that a sample is taken randomly from a population in order to minimize bias. Balancing means that treatments have the same characteristics and reduce variations in the treatment setup to emphasize an effect of interest. Therefore, the number of variables is low compared to survey and case study. Randomization also requires a larger population than a case study. Blocking means that factors which can lead to an effect which is not of interest are deliberately shut out during a treatment. Blocking and a high level of measurement control imply that the analysis of an experiment accumulates measurements over multiple treatments. Thereby, peculiar observations during a single treatment can be lost. Especially, if randomization is impractical, a quasi-experiment evaluation strategy can be applied. In particular, in a quasi-experiment the evaluation objects are not randomly assigned to evaluation subjects. Therefore, the results of a quasi-experiment have less statistical significance than real experiments.

The context of the domain-specific ontology-based model integration poses constraints for the selection of an evaluation strategy. The population of domain experts in conceptual aircraft design is small compared with other expert groups in aircraft design. The APA model, which was used as a sample model during the development and the evaluation, was developed and operated by one person. The case that the population of Model Owners consists of only one person for a particular model is common in the context of conceptual aircraft design. Furthermore, as mentioned in Chapter 1, the design freedom and the number and range of characteristic variables of Integrate Models in this context is high. For instance, each sample model has a different decomposition strategy. Furthermore, the APA and APD sample models have considerably more model objects than the SIMCAD sample model. If these sample models were used in experiments, both kinds of differences would threaten the comparability of the results. Considering the

small population of adequate subjects, an evaluation strategy which requires a large number of subjects is not viable. Accordingly, with respect to the variety of characteristic parameters of the domain context, only typical situations can be evaluated which does not allow for the generalization of the results to the overall field of conceptual aircraft design. Furthermore, the evaluation of the OIDA framework is rather focused on the demonstration of the viability of the solution rather than the generalization of the OIDA framework.

These constraints led to an evaluation design shown in Table 7.1. As the transformation algorithm does not require user interaction, it was evaluated by an automated unit test which gave full control over the testing procedure. As the limited number of sample models did not allow strict randomization of input data, a quasi-experiment evaluation strategy was applied. The Match Models and Merge Models Partially use cases were evaluated in case studies with typical Model Owners such as Model User, Model Developer, and Model Adopter as evaluation subjects. The Ontology Owner was enacted by the observer. The essential components facilitating the Co-Evolution capability do not require user interaction. Therefore, the capability was evaluated by automated unit tests. As the evaluation aimed at the demonstration of a typical yet very small number of Co-Evolve Integration Artifacts use cases, randomization of cases of co-evolution was impractical. Therefore, quasi-experiments were chosen as an adequate evaluation strategy which promised to deliver the desired evidence with sufficient confidence.

For the quasi-experiments as well as in the case studies the APD, APA, and SIMCAD sample models were used as `DisciplineSpecificModels` for model integration. These models are described in Section 3.1. For all quasi-experiments and case studies a prepared `ReferenceOntology` O_R was used, except for quasi-experiment QE 2.2 which evaluates the handling of a modified `ReferenceOntology` to $O_R{}'$.

The evaluation required a prototypical implementation of the OIDA framework. It was decided to implement it in a layered open plug-in architecture based on EMF and the JENA framework. These implementation decisions allowed the integration of the prototype as an extension of the existing OPENCDT application. Basically, the implementation reflects the framework design described in Chapter 6. However, the testing of `KnowledgeSources` revealed that the `NameEquivalenceFinder` and the `SemanticValidator` were ineffective for the given sample models. Therefore, it was decided to implement the `Matching`, `MatchingApp`, and `Evolution` plug-in without a BLACKBOARD pattern as it was not considered to be essential for the evaluation. Appendix A describes the overall implementation of the prototype in detail. The following section addresses only evaluation-specific aspects of the implementation.

7.2. Empirical Studies

In the following, the quasi-experiments and case studies shown in Table 7.1, which are performed to evaluate the OIDA, are described in detail.

Section 7.2.1 describes the quasi-experiments QE 1.1, QE 1.2, and QE 1.3 evaluating the transformation T_{MO} that generates an ontology from a given model.

Capability	Evaluation Strategy	Evaluation Subject	Empirical Study	Evaluation Objects
Transformation T_{MO}	Quasi-Experiment	Transformation subsystem	QE 1.1	APD
			QE 1.2	APA
			QE 1.3	SIMCAD
Model Matching	Case Study	Model User	CS 1.1	APD, O_R
		Model Developer	CS 1.2	APA, O_R
		Model Adopter	CS 1.3	SIMCAD, O_R
Partial Model Merge	Case Study	Model User	CS 2.1	APD, O_R
		Model Developer	CS 2.2	APA, O_R
		Model Adopter	CS 2.3	SIMCAD, O_R
Co-Evolution	Quasi-Experiment	Evolution subsystem	QE 2.1	APD, APA, SIMCAD, O_R
			QE 2.2	APD, APA, SIMCAD, $O_R{}'$

Table 7.1.: Evaluation of the capabilities provided by the OIDA framework to facilitate ontology-based model integration

Section 7.2.2 describes the case studies CS 1.1, CS 1.2, and CS 1.3 evaluating the Model Matching capability of OIDA.

The next Section 7.2.2 describes the case studies CS 2.1, CS 2.2, and CS 2.3 evaluating the Partial Model Merge capability of OIDA.

The evaluations of the Co-Evolution capability in the quasi experiments QE 2.1 and QE 2.2 are described in Section 7.2.4.

The threats to validity of these empirical studies are stated in Section 7.2.5.

7.2.1. Evaluation of the Transformation T_{MO}

The transformation $T_{MO} : M \rightarrow O$ translates a given MOF compliant model to an OWL ontology. The `Transformation` subsystem provides this service to generate an ontology from a discipline-specific model in order to provide the model matching capability. The objectives of the quasi-experiments QE 1.1, QE 1.2 and QE 1.3 are to analyze the transformation of three discipline-specific sample models to ontologies in order to decide whether the transformation is a structure-preserving mapping (homomorphism). Thereby, the analysis of the transformation disregards model and ontology features which are not within the scope of the OIDA framework. These quasi-experiments use the following metrics:

Integrity This measure is the basis for all further measurements. The integrity of artifacts on both ends of the transformation T_{MO} indicates whether the transformation maintains intrinsic integrity which is defined by the validity of models and the consistency of ontologies. The validity of the model as well as the consistency of the ontology can be tested by algorithms which analyze the respective artifact based on the grammar of the modeling language and the ontology language. Model validation is mostly a syntax validation whereas an ontology validation is done by testing the consistency of logical statements using automated reasoning.

Instance Entities The number of model objects at the instance level is compared to the number of individuals in the ontology. Equal numbers of individuals and model objects indicate that the transformation is injective at the instance concept level.

Meta Entities The number of model classes ie compared to the number of ontology classes. Equal numbers of ontology classes and model classes indicate that the transformation is injective at the meta concept level.

Meta Concept Links The number of model associations is compared to the number of ontology object properties. Equal numbers of object properties and associations indicate that the transformation is injective at the meta concept level regarding the mapping of concept links.

Meta Attributes The number of model class attributes is compared to the number of datatype properties in the ontology. Instances of attributes, such as object attributes are not considered. Equal numbers of datatype properties and attributes indicate that the transformation is injective at the metaconcept level regarding the mapping of meta attributes.

Taxonomy Depth The maximal depth of the model inheritance forest[1] is compared to the maximal depth of the inheritance forest of the ontology. The taxonomy of the model and the ontology can have a forest structure as the classes instantiated in the model do not have to be explicitly derived from a single class in the metamodel. Taxonomy depth is an important structural concept of both metamodels and ontologies. Equal taxonomy depths of model and ontology indicate that the transformation preserves the structure of inheritance relations.

Containment Depth The maximal depth of the model containment forest is compared to the maximal depth of an equivalent forest in the ontology. EMF defines Containment as a language construct. As OWL does not specify Containment, the `partOf_directly` object property was defined in the `mereology.owl` upper ontology and was considered to be equivalent to Containment in EMF. The same upper ontology is used in the

[1] In this context a forest is a set of trees. A tree is a set of data organized in a tree structure.

`OntologyGenerator` implementation of T_{MO}. The Containment references of the model as well as in the ontology can form a forest structure as the instances do not have to be explicitly derived from a single instance object. Although all test models have a single root object, the containment depth measure does not determine whether the structure is a tree or a forest. Ontologies can express other kinds of containments. However, the `partOf_directly` is the most important containment concept from the Model Owner's perspective. Equal containment depth indicates that the transformation preserves the Containment structure.

Evaluation-specific Implementation

The algorithms measuring the `disciplineModel` have been realized by methods of the `ModelMetrics` Helper Class. Element counters like `countInstanceObjects()`, `countClasses()`, `countAttributes()`, and `countReferences()` are based on the respective `ModelElementFilters`. Accordingly, the `countClasses()` method only counts classes which are, at least indirectly, used by the measured `disciplineModel`.

The `getContainmentDepth()` and `getTaxonomyDepth()` methods use the reflection capability of EMF.

In an analog manner, the algorithms measuring the `disciplineOntology` have been realized by methods of the `OntologyMetrics` Helper Class. Element counters like `countIndividuals()`, `countOntologyClasses()`, `countDatatypeProperties()`, and `countObjectProperties()` are based on the respective `OntologyFilters`. The `getContainmentDepth()`, `getTaxonomyDepth()` use the ontology reflection capabilities of the Jena framework. The quasi-experiment is implemented as a test program using the JUnit framework [GB99].

Results and Analysis

Table 7.2 shows the measurements collected by the quasi-experiments over all three sample models. Most of the model concepts are translated to the ontology. However, there are discrepancies among all the sample models regarding the number of instances, and between the APD model and its derived ontology regarding the number of meta attributes.

The difference in the number of instances indicates that the renaming schema used in the transformation leads to ambiguous URIs. This finding was supported by warning messages emitted by the unit tests of the `OntologyGenerator` with the sample models. The `RenamerStrategy` used for the transformation was the ContainerName.ObjectName schema described in Section A.4. The selection of this `RenamerStrategy` was based on the assumption that a valid URI can be generated from the container and the concept name. If the renaming creates the same identifier for two semantically different objects, the underlying assumption is invalid.

The limited number of ambiguous renamed objects allowed an object-by-object analysis which revealed different reasons for ambiguity for each source model. In the APD model the objects were ambiguously renamed in the context of the landing gear and the

Quasi-Experiment	Measure	Model	Ontology
QE 1.1	Integrity	Valid	Consistent
	Instances	484 EObjects	458 Individuals
	Meta Concepts	7 EClasses	7 Classes
	Meta Concept Links	1 EReference	1(4) ObjectProperties
	Meta Attributes	3 EAttributes	3 DatatypeProperties
	Taxonomy Depth	6	6
	Containment Depth	7	7
QE 1.2	Integrity	Valid	Consistent
	Instances	466 EObjects	433 Individuals
	Meta Concepts	7 EClasses	7 Classes
	Meta Concept Links	1 EReference	1(4) ObjectProperties
	Meta Attributes	3 EAttributes	2 DatatypeProperties
	Taxonomy Depth	5	5
	Containment Depth	3	3
QE 1.3	Integrity	Valid	Consistent
	Instances	241 EObjects	222 Individuals
	Meta Concepts	7 EClasses	7 Classes
	Meta Concept Links	1 EReference	1(4) ObjectProperties
	Meta Attributes	3 EAttributes	3 DatatypeProperties
	Taxonomy Depth	6	6
	Containment Depth	5	5

Table 7.2.: Measurements of transformations from the APD, APA, and SIMCAD source models to ontologies. EObject, EClass, EAttribute, and EReference are implementations of the respective UML concepts in EMF. The transformation imports the `merelology.owl` upper ontology to represent containment relationships. The number of imported object properties is given in brackets.

tank group of the main wing tank and the horizontal stabilizer. The reason is a reuse of the landing gear and tank group composite structure in several places in the model. The APD Model Owner considered a model modification that would solve the problem to be too complex.

An analysis of the ambiguously renamed objects of the APA model revealed a deep copy of the containment tree under the `geometry` container node in the `configuration` container node. The Model Owner of APA confirmed this observation. Tests with a ContainerContainerName.ContainerName.ObjectName `RenamerStrategy` showed no falsely dropped objects for the given sample models. The renaming schema also dropped one of two attributes called `value` attribute. In fact, the given metamodel stipulates a `value` attribute in both the `IntegerValue` and the `FloatPointValue` class. The chosen `RenamerStrategy` using simply the StructuralFeatureName for attributes and associations dropped one of these attributes. Tests with the given sample models using a modification of the `RenamerStrategy` to ContainingClassName.attributeName showed no false dropping of attributes.

A closer analysis of the SIMCAD model revealed two reasons for ambiguous mappings, namely a bug in the OPENCDT tool connector, and a design principle of the SIMCAD model. Due to a bug in the OPENCDT tool connector the algorithm mistakenly creates two `mass` container objects. The original SIMCAD data model file does not contain these doublets. Similar to APD, SIMCAD uses structural copies of the model tree to represent different parts of the aircraft mission by the same parameters. Both cases of ambiguity in SIMCAD are not examined further in this evaluation as their origins are beyond the scope of the OIDA framework. However, the ambiguities in the APA and the APD sample models were considered problematic and chosen for an exemplary use case of co-evolution due to modifications of the model integration framework and the reference ontology modification. In order to create a base line, the case studies evaluating the Model Matching and Partial Model Merge capabilities were conducted without modification to the renaming algorithm and the reference ontology.

7.2.2. Evaluation of the Model Matching Capability

The objective of evaluating the Model Matching capability of OIDA is to analyze the performance and the oral feedback of Model Owners during the matching process of his aircraft model in order to determine the perceived efficiency of the Match Models use case. The Model Matching capability was evaluated using the following measures:

Duration of Mapping This measure is the time span it takes the Model Owner to map the elements of his discipline-specific model to the reference ontology. The time necessary to get used to the user interface is not counted.

Mapped Model Objects During the case study the Model Owner maps objects from his discipline-specific model to the `referenceOntology` by equivalence statements. This measure is given together with the total number of mappable model elements.

Perceived Efficiency After the treatment, the Model Owner is asked by the observer to rate the efficiency of the mapping task. The observer defines efficiency as the cost of time spent for model matching versus the benefit of model integration. The rating scores are defined using an ordinal qualitative scale as shown in Table 7.3. On this scale a score greater than 4 indicates that the task efficiency was unacceptable for the Model Owner.

			← Acceptable	Unacceptable	→
1	2	3	4	5	6
Very good	Good	Satisfactory	Sufficient	Insufficient	Fail

Table 7.3.: The ordinal scale of the task efficiency perceived by the Model Owner

Evaluation Objects

A separate sample model was used for each case study. Therefore, the `ToolConnectors` of OpenCDT were modified to disregard non-scalar values. The reference ontology which is described in detail in Appendix B was also prepared in advance in Protegé. This reference ontology was used without modifications for all cases studies.

Evaluation Subjects

The evaluation subjects were selected according to the following criteria:

- They are experts in conceptual aircraft design and have already conducted several conceptual aircraft design projects.
- Before the case studies they were not familiar with the Oida framework.
- They are familiar with aircraft modeling due to their experience as users or developers of conceptual aircraft design tools.

Case Study Design

Each case study is structured as a sequence of the four phases, introduction, user interface familiarization, concept mapping, and debriefing.

During the *introduction* the observer instructs the Model Owner with respect to the objective of the study. Furthermore, the observer states that model elements representing behavioral aspects of the system and tool-specific concepts are intentionally excluded from the reference ontology as they are not within the scope of the case study. Before the actual mapping phase, the Model Owner is asked to identify the given sample model by the `ModelNavigator`. Then the observer, enacting the Ontology Owner, begins the *user interface familiarization* by demonstrating the mapping of metamodel concepts to the `referenceOntology`. The Model Owner is informed that the actual mapping process

is limited to about 30 min and encourages him to provide feedback during the task about annoyances or ideas for improving the user interface. Then the actual *concept mapping* starts. After about 30 min the observer can either abort or continue the mapping process for further 15 min. In the *debriefing* phase which is performed directly after the mapping process, the collected feedback is summarized by the observer for clarification purposes. The observer then categorizes the feedback into accidental issues and substantial issues. Accidental issues are related to the current state of the implementation or the content of the reference ontology which can be addressed by evolutionary refinements to the OIDA framework. Substantial issues address basic assumptions, design decisions, and implementation of the OIDA framework. Only substantial issues are considered further.

Evaluation-specific Implementation

The user interface of the `MatchingApp` application prototype was implemented specifically to facilitate the evaluation of the Model Matching capability.

The `MatchingEditor` depicted in Figure 7.1 supports the user to match a given `disciplineModel` to concepts in the reference ontology by declaring equivalence relationships. It is based on the `ExtendingMatchingEditor` which allows the Model Owner and Ontology Owner to choose a `disciplineInstanceObject` or `disciplineMetamodelElement` and the semantically equivalent `referenceOntologyResource`. However, the actual version of the `Editor` used in all case studies is deprived of the capability to extend the `referenceOntology`.

The left hand side of the editor shows the `Individuals` and `MetaConcepts` such as `OntologyClasses` and `Properties` of the ontology derived from the current `disciplineModel`. The right hand side shows possible `Individuals` and `MetaConcepts` from the reference ontology for a selected element of the `disciplineOntology`. The UI filters the candidate reference concepts based on declared equivalences and existing object properties in the ontologies. With this filter, the Model Owner can find an appropriate reference concept more quickly and inconsistent equivalence mappings are prohibited. The current implementation only takes the OWL-defined Properties `instanceOf` and `subClassOf` into account. For instance, if the user wants to map `nPax` he can first classify it as `CounterOfPersons`. This user action triggers filtering all `Individuals` to eight candidates which are instances of `CounterOfPersons`. The user clicks on the "Confirm" button to trigger the creation of the equivalence relationship. If an equivalence is confirmed, the status of the ontology is reevaluated and the button changes to "Revoke". This allows the user to undo equivalence mappings. As the `InferenceModel` was not integrated, its delaying effect of the `Reasoner` was simulated by a user interface delay of about 3 s. The result of the status evaluation is shown in a progress report. The current implementation displays the number of unmapped concepts. A mock widget mimics the `Reasoner` status when the `InferenceModel` is fully integrated.

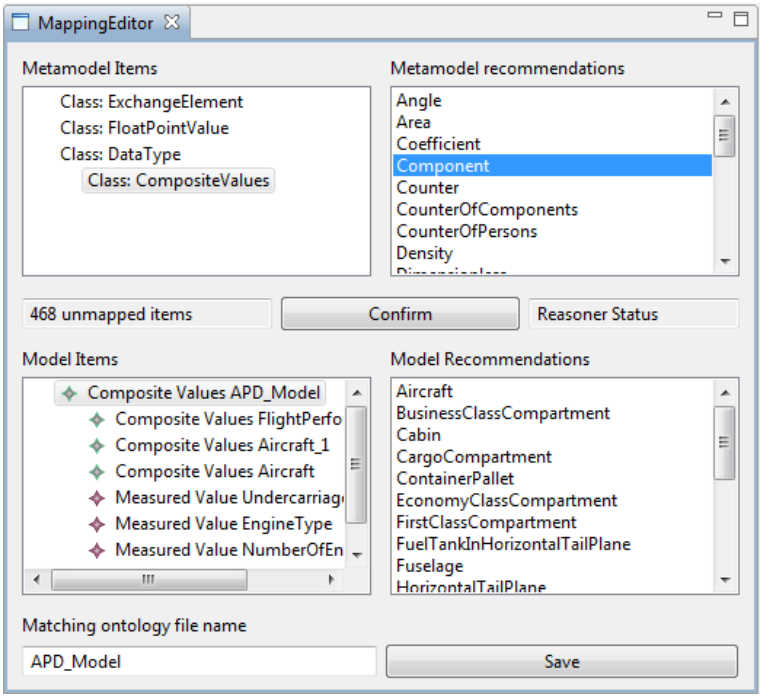

Figure 7.1.: A screen shot of the `Matching Editor` during the matching process of the APD aircraft model

Case Study Execution

CS 1.1 Model User The evaluation subject in this case study is an aeronautics engineer who is experienced in conceptual aircraft design in general and in using the APD tool in particular. He was, however, not engaged in developing the APD tool, which qualifies him as a Tool User.

The introduction and GUI familiarization took 10 min. During the user interface familiarization the Tool User confirmed that the model was an APD model.

During the process he commented on substantial issues addressing the integration of the reasoner and the interactive modification of the reference ontology. The interaction delay after each mapping which simulated the reasoner was perceived as very annoying. He proposed performing the check not after each mapping but on the user's demand.

After 33 min the observer decided to stop the mapping process. At this point, the Model User had been able to map 45 of 458 possible model elements. During the debriefing the Model User rated the task efficiency as "satisfactory".

CS 1.2 Model Developer In the second case study, the evaluation subject was the Model Developer of APA who also frequently uses APA as a tool. Furthermore, the same initial version of the `referenceOntology` was used as in CS 1.1.

The introduction took 9 min. Before the familiarization process, the Model Developer had to delete parts of the model which were not within the scope of this case study, e.g., simulation logs or non-scalar data structures. However, the Model Developer confirmed that the integrity of the APD model regarding structural aspects of the aircraft were still intact. In the next part of the case study, the observer demonstrated the user interface by mapping the metamodel. In this case only two attributes could be mapped. All other entities had no equivalent concept. During the next 34 min, the Model Developer was able to map 13 concepts. Thereby, the observer noted that about 80 % of the concepts of the APA model did not exist in the reference ontology.

During the concept mapping phase, the Model Developer commented on issues addressing the interaction design of the mapping task. The dynamic type casting of MATLAB, where the APA sample model originates, can disturb the identity of model objects within the context of the `MatchingApp` application. In MATLAB, the object identity does not change due to dynamic type casting. However, if a dynamically changed data type is propagated to a JAVA runtime environment, the object identity changes completely. Furthermore, assumptions about the semantics of data types can be misleading, for example, an item was casted as `Integer` but does not represent a counter. He also suggested that the software could take the order of magnitudes of attribute values in into account to recommend matching individuals. Furthermore, he missed a feature which would allow the actors to add concepts to the `referenceOntology`. This feature was deliberately disabled during all case studies. He considered the presence of the Ontology Owner to be helpful for avoiding false equivalence statements or for finding appropriate reference concepts more quickly. The Model Owner did not perceive the delay by the simulated `Reasoner` as an annoyance and rated the perceived task efficiency "satisfactory" to "sufficient".

	CS 1.1	CS 1.2	CS 1.3
Evaluation Subject	Model User	Model Developer	Model Adopter
Evaluation Object	APD, O_R	APA, O_R	SIMCAD, O_R
Duration of Mapping	33 min	34 min	37 min
Mapped Model Concepts	45 of 458	13 of 433	66 of 222
Perceived Efficiency[a]	3	3 – 4	2

[a] See scale on Table 7.3

Table 7.4.: Summary of the case studies evaluating the Model Matching capability of the OIDA framework

CS 1.3 Model Adopter This case study addressed the situation when a Model Owner was not familiar with a given discipline-specific model which he wanted to adopt, but could use his expert knowledge to map this model to the reference ontology. In this case, the evaluation subject had the role of a Model Adopter. The chosen person was familiar with aircraft design tool development in general and with APD in particular, but not with SIMCAD which was the object of this case study. As the Model Adopter was not familiar with the source model, he could not confirm the authenticity of the source model but recognized it as a conceptual aircraft model.

After the short introduction to the user interface by the observer lasting about 10 min the Model Adopter was able to map 66 entities in 37 min. Thereby, he used not only the object names but also the attribute values of the objects to guess their meaning. The Model Adopter could also use the names of the composite objects to determine the context of the leaf objects containing aircraft attributes.

During the process, the Model Owner addressed the following substantial issues: The potential benefits of a consistency check by the Reasoner after every transaction and recommendation generation was considered beneficial. Accordingly, the simulated delay by the `Reasoner` was not perceived as an annoyance. Units of measures and orders of magnitude of numeric attribute values could be used for equivalence recommendations. The Model Owner rated the task efficiency as "good".

Analysis

The data collected from the case studies and presented in Table 7.4 show that all subjects comprehended the concept of semantic mapping and were able to perform their tasks. All evaluation subjects required only a short introduction and assistance by the Ontology Owner enacted by the observer.

The mapping efficiency seems to depend strongly on the overlap between the concepts of the respective discipline-specific models and the concepts in the reference ontology. If a considerable number of concepts of one discipline-specific model are not contained in the reference ontology which has been derived from another discipline-specific model,

both models are either on different levels of detail or have a small semantic overlap. Therefore, it is important to provide a reference Ontology which covers most of the concepts contained in the discipline-specific models.

Especially the third case study indicated that for the performance the Match Models use case the Model Owner does not necessarily have to be familiar with the discipline-specific model in order to perform the mapping. All evaluation subjects suggested exploiting the attribute values for equivalence recommendations. Especially the `unit` attribute was identified as an effective indicator for equivalence.

The results of the first and the third case study are very similar, though the scale of the models and the relationship of the Model Owners to their respective discipline-specific model are different. Both the APD and SIMCAD models cover the overall aircraft system without focusing on a specific system, whereas the APA model is focused on propulsion system architectures. As the Model Developer of APA could map considerably fewer concepts to the reference ontology, the case studies could not show that the current implementation can bridge between models having a different focus.

7.2.3. Evaluation of the Partial Models Merge Capability

During the evaluation of the Partial Model Merge capability, the Model Owners are presented with conflicts and import candidates. Both have been determined by matching the discipline-specific model which they are responsible for with the Chief Engineering Model. Both models have been previously mapped to the reference ontology. The Model Owner will not gain the full benefit of the OIDA framework before this Merge Models Partially use case. Comparing conflicting values allows them to assess the effectiveness of the integration framework.

The objective of the case studies evaluating the Partial Model Merge capability is to analyze the feedback of the Model Owner regarding the perceived plausibility of results. In these case studies, the following measures were used which correspond to the terminology of the OIDA design described in Chapter 6:

Semantic Matches The number of object pairs from the discipline-specific source and target model which could be matched by OIDA.

Conflict Candidates The number of semantically matching object pairs which have conflicting states detected by OIDA.

Confirmed Conflicts The number of conflict candidates which have been confirmed by the Model Owner.

Reported Conflicts The number of conflict candidates which have been reported by the Model Owner due to suspected false matchings.

Import Candidates The number of objects from the discipline-specific source model which OIDA could not match with an object in the discipline-specific target model.

Confirmed Import Candidates The number of import candidates selected by the Model Owner for import into his discipline-specific target model.

Reported Import Candidates The number of import candidates which have been reported by the Model Owner due to suspected false matchings.

Perceived Overall Efficiency The efficiency of the Match Models and Merge Models Partially use cases from the Model Owner's perspective.

Result Plausibility The plausibility of the proposed conflict and import candidates with respect to the method applied by the OIDA framework and conveyed by the observer.

Evaluation Subjects

The evaluation subjects of these case studies were the same as in the previous case studies. This implies that they were already informed about the overall procedure and context of the integration process. As with the previous case study, the observer enacted the role of the Ontology Owner.

Evaluation Objects

In these case studies, the same APD and APA sample models were used as in the previous case studies. The SIMCAD model was declared both the Chief Engineering Model, which overrides the APD and APA model in the case of conflict, and the source model of Migrate Model Parts. Furthermore, the ontologies containing the equivalence statements created in the first phase of the case studies were used.

Case Study Design

The case studies addressing the Partial Model Merge capability evaluated the plausibility of the matching results perceived by the evaluation subjects. Furthermore, at the end of the Merge Models Partially use case, the conflict resolutions are propagated to the discipline-specific target model and the ontologies are checked for consistency with this changed target model. The case studies consist of the following phases: introduction, conflict resolution, partial model import, and debriefing.

In the *introduction* phase of the case study, the Ontology Owner explains the task to the Model Owner. The Model Owner is instructed that he will not be able to override incoming changes but has to assess whether the incoming values make sense. It is further agreed that Model Owner and Ontology Owner can talk freely about the user interface and presented conflict and import candidates. The Model Owner is also informed that the time until completion of the task is recorded.

During *conflict resolution*, the Model Owner is presented with a list of conflicting objects. He sees both object names from the target model and the reference ontology but is not provided with the object name from the source model. The only way he

can assess the plausibility is the order of magnitude of numerical attribute values and, if available, the `unit` attribute. The Model Owner confirms the plausibility of each conflict or explicitly declines it. The Model Owner completes the conflict resolution, which triggers the transformation of the result to the discipline-specific target model. Subsequently, the Model Owner has to confirm that the conflict resolution has preserved the nomenclature and structure of the discipline-specific target model.

At the beginning of *partial model import*, the user interface provides a list of model objects of the incoming model which could not be matched with the target model. The Model Owner can decide which objects he wants to import and chooses an object name and a container appropriate to his own naming convention and model decomposition strategy.

In the *debriefing* phase, the Ontology Owner wraps up the feedback which was collected during the session. The Model Owner is further asked whether he considers the matching result plausible and whether the preceding phases of the case study were perceived to be useful in the retrospective on the scale shown in Table 7.3

Case Study Execution

Based on the results of the evaluation described in Section 7.2.2 the following case studies are executed when Model Owners want to check whether their respective model has conflicts with the `ChiefEngineeringModel`, and to select objects for import from that model.

CS 2.1 Model User

This case study was performed with the same evaluation objects and subjects as in CS 1.1. First, the Model User confirmed that the model represented an APD model.

It required about 5 min of introduction by the observer before the Model Owner could begin with the conflict resolution using the `ImportCandidateReviewForm`. The Model User confirmed 14 conflicts and reported 4 of 18 conflicts, because the incoming values appeared to be undefined. Then the Ontology Owner proceeded to the `MigrateDialog`. After a short introduction by the observer, the Model User confirmed no import candidate and reported 28 of 47 import candidates. The Model User explained that the SIMCAD model appeared to provide more details on aircraft cabin design than the APD model. In its own work flow, the cabin interior was designed in a separate CAD application. He also stated that importing data on demand was effective to manage the complexity. He was sure that all incoming values he reported were already in the APD model but were not mapped to the reference ontology in CS 1.1. Another Match Models session would be required to completely map the APD model to the reference ontology. The conflict resolution and migration parts were concluded after 17 min. In the debriefing, the Model User gave the matching process an efficiency rating of "good", given that the OIDA framework provides the Co-evolution capability. He also mentioned, that the mapping procedure could be an error-prone task. Therefore, the number of mappings should be reduced as much as possible. Generally, he considered the results of the

matching process plausible. The reported conflicts and SIMCAD data elements seemed to him more the result of mapping error by another Model Owner than an error by the system.

CS 2.2 Model Developer The introduction phase took 5 min. At the end of the introduction, the Model Developer recognized the APA sample model as his model. During the conflict resolution phase the Model Developer criticized that the user interface presents the model elements without their context which makes it difficult to comprehend the meaning of the elements. For instance, the thrust of an engine has almost no meaning without setting it in the context of the velocity, pressure and temperature of the currently surrounding atmosphere. The Model Developer commented on the character of the values coming from the Chief Engineering Model. For instance, he assessed the empennage as an aggressive design. The Model Developer confirmed all presented conflicts. The conflict resolution phase ended after 8 min. During the model migration phase he noted that the units of the incoming system attributes seemed to match his implicit convention for units of attributes in the APD model. However, further implementations should provide automated unit conversion or a generic conversion factor. He was sure that most of the import candidates were already in his model but were not mapped during CS 1.1. Accordingly, he reported most of the import candidates as errors. However, he imported two model elements into his structure using the user interface to create the appropriate container model element in his model. The migration phase ended after 10 min. In retrospect, he perceived the efficiency of the process as better than in the first case study C 1.2 and rated it "good" to "satisfactory". He also considered the result plausible.

CS 2.3 Model Adopter In this case study the model owner enacted the role of a user who wants to import a model from SIMCAD to APD. During the introduction he identified the APD sample model using the `ModelNavigator`. The introduction took about 5 min. The Model Adopter had not mapped the APD model himself. Therefore, he had to trust another Model Adopter to have performed the mapping correctly. The user interface did not help him to achieve an overview of the model structure as the conflicts are provided as a list. He only reported one value which, in his opinion, was obviously mapped wrongly by the APD Model Owner. The conflict resolution phase ended after 15 min. The Model Adopter started the migrate dialog. He imported 7 of 47 import candidates which he considered relevant for the scope of APD and used the user interface to declare a suitable container for the new elements and to assign suitable names. Thereby, he commented that it would improve the efficiency if the nomenclature from the reference ontology could be taken over automatically. In some cases, the Model Owner chose to confirm model elements whose attribute values were not set. He expected the values to be set in future iterations, and wanted to "subscribe" to these model elements. He reported 8 of 47 proposed import candidates because he was sure that they were already in the discipline-specific target model but not mapped by its Model Owner during CS 1.1. In his opinion, the user should be able to map a proposed

item to the existing model elements instantaneously. The migration phase ended after 20 min. In retrospect, the Model Adopter confirmed his original rating of a "good" overall efficiency. In his opinion, the system generated plausible results. Furthermore, he considered the migration of model elements on demand very helpful and requested that future versions of the user interface should provide a more differentiated means of reporting.

Analysis

During the second group of case studies several general observations could be made. One part of the observations refers to the validation of the Partial Model Merge capability which was evaluated with these case studies. The other part of these observations refers to the overall process associated with ontology-based model integration. All case studies in the second group supported the observation that all Model Owners could operate the prototype with very little assistance from the Ontology Owner. Regarding the overall process, the second group of case studies made the subjects change the rating of their perceived efficiency for the better or maintain an already positive rating. Furthermore, all Model Owners considered the results as plausible. The reported problems were seen as mistakes of mappings by a Model Owner but not as a dysfunction of the OIDA implementation. The man-in-the-loop strategy of the Model Matching and Partial Model Merge capability of OIDA was also perceived to be very useful. Especially the APA model revealed the importance of a sufficient minimal content of the `referenceOntology`. Consequently, the APA Model Owner regarded the implemented prototype as not fully applicable to the class of problems he was dealing with.

7.2.4. Evaluation of the Co-Evolution Capability

In Chapter 4.3, typical use cases are described which require co-evolution of coupled artifacts. The objective of the evaluation is to test the Co-Evolution capabilities of OIDA in different use cases of Co-Evolve Integration Artifacts. The Co-Evolution capability of the OIDA framework to support the Evolve due to Model Integration use case could be verified after the case studies described in Section 7.2.3 by manual inspection in OPENCDT and PROTÉGÉ, respectively.

The evaluation was conducted as two quasi-experiments using automated unit tests each covering two model evolution scenarios. The evaluation of Co-Evolution is based on the results of the previous case studies, evaluating the Model Matching and Partial Merge capabilities. Thereby, the following measures were obtained:

Obsolete Mappings An existing equivalence statement which cannot be redirected automatically by the co-evolution capabilities of the OIDA framework is declared obsolete. Obsolete mappings indicate how much previous effort by the actors involved in the Match Models use case has been lost.

	CS 2.1	CS 2.2	CS 2.3
Evaluation Subject	Model User	Model Developer	Model Adopter
Evaluation Object	APD	APA	APD
Chief Engineering Model	SIMCAD	SIMCAD	SIMCAD
Semantic Matches	18	6	18
Conflict Candidates	18	6	18
Confirmed Conflicts	14	6	17
Reported Conflicts	4	0	1
Import Candidates	47	59	47
Confirmed Import Candidates	0	2	7
Reported Import Candidates	28	0	8
Perceived Overall Efficiency[a]	2	2 – 3	2
Result Plausibility	yes	yes	yes

[a] See scale in Table 7.3

Table 7.5.: Measurements from the case studies evaluating the Partial Model Merge capability of the OIDA framework

Mappings Required by the Model Owner The number of manual actions required by the Model Owner to make the previously mapped ontology consistent with the new version of the model. It is only concerned with revising obsolete and creating new equivalence statements from discipline-specific instance concepts to the reference ontology. This implies that the Model Owner is usually not concerned with mapping concepts from the metamodel. This measure can be correlated with the Obsolete Mappings measure. Obsolete mappings have to be discarded or remapped manually by the Ontology Owner or Model Owner in order to maintain the same level of semantic coupling that existed before the evolution. More Required Mappings by Model Owner than Obsolete Mappings indicate that the modification created new discipline-specific instance concepts.

Creation of Reference Concepts A number of reference concepts need to be created in order to make the reference ontology consistent with the evolved models. In the current concept of Match Models, this task is performed in a collaboration between Model Owner and Ontology Owner. Therefore, this measure indicates the effort required from both actors.

Mappings Required by the Ontology Owner The number of mappings which make the evolved ontologies and models consistent at the metaconcept level. According to the interaction design of the OIDA framework, the Ontology Owner is responsible for this task. Therefore, this measure indicates the effort required from the Ontology Owner.

Evaluation-specific Implementation

The Partial Model Merge capability of the OIDA framework is realized in `MergeControl` of the `Merge` plug-in. The supporting capabilities for the other Co-Evolve Integration Artifacts use cases are implemented in the `Evolution` plug-in. The two classes `RenameReference` and `ImportPreviousMappings` implement two kinds of `KnowledgeSources`.

The `RenameReference` addresses the Evolve due to Reference Ontology Maintenance and implements the `EvolutionAlgorithm` as the implementation is based on the assumption that there is exact knowledge of the modification, the artifacts, and the couplings between them is known. In particular, this class realizes a support for a RENAME refactoring of the `referenceOntology`. If an `OntologyResouce` in the `referenceOntology` is renamed, not only the `Properties` pointing to that entity within the `referenceOntology` but also equivalence statements from `disciplineOntologies` have to be updated. As the BLACKBOARD pattern was not implemented, `RenameReference` was realized as a preprocessor for the Match Models use case which could be tested separately from the `Matching` plug-in. `RenameReference` requires an old `referenceOntology` and a list of old `disciplineOntologies` which contain equivalence statements to the old `referenceOntology`. Then it performs the RenameRefactoring to `Properties`, `Classes` and `Individuals`. Equivalence statements which become obsolete are discarded and

counted for measurement. Successfully adjusted old equivalence statements are automatically created in the regenerated new versions of `disciplineOntologies`.

The `ImportPreviousMappings` class addresses all use cases of model co-evolution except the first, based on the assumption that the exact modification to the coupled artifacts is unknown. Accordingly, it implements the `EvolutionHeuristic` interface. Equivalence statements are the most expensively created objects of the ontology-based model integration approach with respect to the required work time of the Model Owners. Therefore, the implementation of `ImportPreviousMappings` focuses on the task to import as many previously made `ImportPreviousMappings` as possible, regardless of which kind of modification has taken place. As the BLACKBOARD pattern was not implemented, `ImportPreviousMappings` was implemented as a preprocessor to `MatchingControl` and was integrated with the user interface demonstrator `ExtendingMatchingEditor`. Given a newly generated `disciplineOntology` and a new `referenceOntology` and the respective previous versions, the implemented heuristic first extracts equivalence statements from the old versions. Only if both ends of such equivalence statements can be unambiguously correlated to a new concept, they are recreated in the newly generated `disciplineOntologies`, otherwise are they discarded and counted as obsolete equivalence statements.

Quasi-Experiments

The quasi-experiments for the evaluation of co-evolution are realized as unit tests based on the `disciplineOntologies` which have been generated from the sample models and mapped by the original Model Owners in CS 1.1, CS 1.2, and CS 1.3. The measurements of Obsolete Mappings are provided by `ImportPreviousMappings` and `RenameReference`. The other measurements result from the use of matching and reference creation methods in the respective unit test implementation.

QE 2.1: Co-Evolution due to modeling This automated test stages the situation after an initial integration process when the Model Owner modifies his discipline-specific model by changing the name of an object and by creating a new object. Thereby, the performance of the OIDA framework regarding the Evolve due to Aircraft Modeling use case can be measured. The test executes the Match Models use case and automatically performs the required actions for Co-Evolution and validates the consistency. Thereby, the measurements relevant for the Co-Evolution capability defined above are acquired.

In particular, the automated test changes the name of the object representing the wing span to full wing span in the appropriate nomenclature of the respective model which represents a Rename refactoring. Then a new model element is created which represents the maximum cabin pressure. The test run is performed with each sample model separately.

QE 2.2: Algorithm and Reference Ontology Modification The objective of this quasi-experiment is to determine how much effort is required for the co-evolution of all artifacts

due to Evolve due to Reference Ontology Maintenance and Evolve due to Integration Software Maintenance.

The evaluation scenario addresses an issue discovered during the evaluation of the transformation T_{MO} described in Section 7.2.1. The analysis of the `OntologyGenerator` showed that not all attributes contained in the APA metamodel were transformed to the reference ontology. The reason for this unintended behavior was the `RenamerStrategy` employed for the evaluation which generates the URI of the `OWL:DatatypeProperty` and `OWL:ObjectProperty` using the `name` attribute value of `EAttribute` and `EReference` defined by EMF. For the given metamodels, this `RenamerStrategy` was ambiguous. A solution for this problem required two modifications: First, the renaming algorithm was changed to add the value of the respective name of the `ContainingClass` provided by EMF to the URI of the DatatypeProperty and ObjectProperty. Second, the first version of the `referenceOntology` was extended by two new DatatypeProperties `hasRealNumberValue` to the XSD type `BigDecimal` as a subproperty to the existing `hasValue` and `hasIntegerNumberValue` to the XSD type `BigInteger` as subproperty of the `hasRealNumberValue`. The range of the existing `hasValue` was removed. This modification of the `referenceOntology` was conducted in Protégé as it required specification of domain, range, and super-property, which is not yet supported by the `ExtendingMatchingDialog` of the Oida framework. These modifications represent a maintenance of both the reference ontology and the Oida implementation.

The implemented unit test starts the Match Models use case with the new modified transformation algorithms and the modified `referenceOntology` and imports the old mappings from the status before the modification. Then the automated unit test performs the required actions of co-evolution automatically and validates the consistency. Thereby, the measurements relevant for co-evolution defined above are acquired. The unit test is performed with each sample model separately.

Analysis

The quasi-experiment QE 2.1 shows that the Oida implementation creates consistent artifacts after the integration process. Furthermore, most of the previously created equivalence statements can be adjusted automatically.

In particular, the results of QE 2.1 given in Table 7.6 show that the Co-Evolution capability is supported by the Oida framework. In this use case, small modifications require only a small number of actions by the Model Owner. Due to the Rename Refactoring, the identifier changed which led to one obsolete mapping which had to be remapped by an action typical for the Model Owner. The new element required the addition of new Individuals to the reference ontology which required the attention of both the Ontology Owner and the Model Owner. No particular mapping action was required from the Ontology Owner.

The results of experiment QE 2.2 in Table 7.7 show that the modifications of the reference ontology and the renaming algorithm require only a small number of actions by Model Owner and Ontology Owner to maintain the same level of mapping between the source models and the reference ontology as the Model Owner achieved in the previous

		APD	APA	SIMCAD
Changes	New model elements	1	1	1
	Modified model elements	1	1	1
	New reference concepts	0	0	0
Actions	Obsolete mappings	1	1	1
	Mappings by Model Owner	2	2	2
	Creation of reference concepts	1	1	1
	Mappings by Ontology Owner	0	0	0

Table 7.6.: The modifications versus the required actions by the Ontology Owner and Model Owner observed in the automated test QE 2.1

		APD	APA	SIMCAD
Changes	New individuals	26	33	18
	New classes	0	0	0
	Creation of reference concepts	2	2	2
Actions	Obsolete mappings	0	4	0
	Mappings by Model Owner	0	6	0
	Creation of reference concepts	0	0	0
	Mappings by Ontology Owner	0	2	1

Table 7.7.: The modifications versus the required actions by the Ontology Owner and Model Owner observed in the automated test QE 2.2

Match Models use case. Almost all equivalence statements could be imported automatically. Only the mapping of the `hasValue` property was declared obsolete as the source of the object property did not exist anymore in the newly derived ontology. The Ontology Owner only had to map the new object properties `IntegerValue:value` and `DoubleValue:value` to `hasIntegerNumberValue` and `hasRealNumberValue`, respectively. No new concepts had to be added to the reference ontology. Subsequent tests showed that the `Merging` plug-in can handle the situation in which `Counters` like `nPax` which had been mapped to `NumberOfPassengers` had a `value` attribute as `Double` data type in the source model but an `Integer` attribute in the reference representation of the model.

7.2.5. Threats to Validity

Due to the following threats, the validation obtained from this evaluation provides only anecdotal evidence. The small sample size of the evaluation is too small to generalize the conclusions to other scales of model complexity and other combinations of engineering disciplines. Furthermore, the three professionals were voluntary, that is, we used convenience sampling. Due to the very limited availability of test models and qualified Model Owners, the subjects were also not randomly selected. For the APA model, there is only one developer and user. Therefore, the conclusions might also be affected by bias. There is a possibility that experimental conditions, namely the different overlappings between sample models and the reference ontology, could make a difference on how participants perceive the tool they had to use. The participants of the experiments were aware that their work was being measured. Therefore, there is a chance that the participants presented a modified or improved behavior. Based on the Hawthorne effect [Ada84], qualities measured in the study could be more than those of real world.

7.3. Results

Regarding the objectives of the evaluation stated before, the following results were attained:

Regarding model and metamodel features which are considered most important for the Model Owners of the sample models, the transformation is structure-preserving (homomorphism). The evaluation also revealed that for some metamodel and model elements, the transformation was ineffective in generating unique ontology resource identifiers. This issue was addressed by an instance of an Evolve due to Integration Software Maintenance use case which resolved the issue for the particular sample models. However, the issue illustrates the importance of evaluating the effectiveness of Oida transformations for a concrete application.

The preparation of the case studies CS 1.1, CS 1.2, and CS 1.3 evaluating the interactive Model Matching capability revealed the ineffectiveness of the `NameEquivalenceFinder` and the `SemanticValidator` for the given sample models. Therefore, the case studies were performed on a prototype which only provides the manual matching capability without using a Blackboard pattern infrastructure. Due to time restrictions of the case studies the sample models could not be matched completely against the

reference. However, in the case studies CS 2.1, CS 2.2, and CS 2.3 evaluating Partial Model Merge capability, the OIDA prototype detected 18 conflicts between APD and SIMCAD and 6 conflicts between APA and SIMCAD and successfully imported all selected unmatched model objects. Due to the incomplete matching to the reference ontology, not all conflicts could be identified. Accordingly, all case studies showed that the interaction design of the OIDA framework is effective in facilitating ontology-based model integration. In particular, the feedback from the Model Owners during the case studies indicates that the work flow of the OIDA applications was understood quickly. The Model Owners considered the work flow sufficiently efficient and the produced results such as conflict and import candidates to be plausible.

The case studies also revealed that `ReferenceOntology` should contain most of the concepts used in the `DisciplineModels` before the Match Models use case is started. Furthermore, the `MatchingApp` application should provide the capability to extend the `ReferenceOntology`. The latter was deliberately disabled during the case studies.

The quasi-experiment evaluating the Co-Evolution capability showed the effectiveness of both heuristic and algorithmic approaches in the OIDA framework for exemplary instances of the Co-Evolve Integration Artifacts use case. In particular, in all quasi-experiments of QE 2.1 and QE 2.2, the OIDA prototype always required an equal or lower number of actions compared to the number of changes between artifact versions. Despite this successful proof-of-concept, the current implementation did not allow the evaluation of Co-Evolution `KnowlegeSources` within the context of the envisioned BLACKBOARD pattern.

8. Conclusion and Outlook

Ontology-based model integration addresses the problem of inconsistencies between the many aircraft models that are created in a conceptual aircraft design process, and the inefficient migration of model elements between discipline-specific models. In this dissertation, these problems have been analyzed using three typical aircraft concept models from different provenience, scope and level of detail. Based on this analysis, the OIDA framework has been designed to provide the ontology-based model integration capability engaging the Model Owners in an interactive process. This process and a prototype of the OIDA framework have been evaluated by quasi-experiments and case studies using real conceptual aircraft models as sample models and the respective Model Owners as participants. In this evaluation, the OIDA prototype demonstrated conflict resolution and migration of selected model elements between the sample models.

The analysis and the development of OIDA was focused on the integration of structural aspects of the aircraft concept models. Further research is required to investigate whether ontology-based model integration can be applied to more aspects of an aircraft model or even to other domains of engineering.

This dissertation shows how *ex post* semi-automated application of formal semantics to existing conceptual aircraft models and their evaluation enables model matching across different naming conventions and decomposition strategies, the identification of conflicts between equivalent model elements, and the selective migration of model parts. Presumably, an earlier integration of formal semantics in models and metamodels and a deeper integration of semantic technologies into the design work flow will not only improve the efficiency of communication between heterogeneous information systems but also facilitate better human-machine interfaces in computer-aided design.

8.1. Contributions

The OIDA framework described in this dissertation was developed and evaluated addressing the research questions stated in Chapter 1. The following contributions were made:

Analysis of Conceptual Aircraft Models Three sample models were analyzed in order to observe and classify differences between conceptual aircraft models which cause a lack of efficiency and effectiveness in existing model merging techniques. The sample models were created in tools currently used by industry and academia for conceptual aircraft design. The sample models represented the same aircraft type but were created with different scope and focus. The parts of the sample models representing structural

aspects of the aircraft were transformed to the same metamodel using existing `ToolConnectors` provided by OpenCDT. Thereby, the transformation allowed manual and automated inspections on a common syntactic basis. Despite the common metamodel, the sample models revealed considerably different naming conventions stemming from different coding styles of the developers and users of the sample models. Furthermore, the sample models exhibited different decomposition strategies. This indicates a gap between the abstraction level of the metamodel and the application domain which prohibits the automated identification of objects representing the same physical object by their properties. Accordingly, a more concrete modeling standard would facilitate finding a solution. However, especially a unification of the design-process-oriented decomposition limits the designers in organizing their models according to their specific methodology. Furthermore, conceptual designers explore new component layouts and system configurations which are not standardized for their domain.

These insights have not only been the design driver for the Oida framework but can also serve as a basis for alternative solutions. However, this analysis disregards parts of the sample models which represent behavioral aspects of the aircraft system. Future work in the context of model integration in conceptual design should examine these aspects which are especially relevant for the design and assessment of operational capabilities of an aircraft.

Ontology-based Model Integration Conventional model merge techniques have a number of shortcomings. To compensate for these the Oida framework was developed also with view to transparency and user control.

In order to compensate for the shortcomings of conventional model merge techniques, the Oida framework was developed. The Oida framework is designed towards transparency and user control. Therefore, it engages the Model Owner who is the persons responsible for a given conceptual model to declare the meaning of model and metamodel elements by equivalence statements.

Instead of using a central all-comprising model as a reference, the solution employs an ontology which provides reference entities. Thereby, the ontology language allows formal definition of semantics of the reference concepts formally by logical statements. The interactive integration process has two phases: In the first phase, each Model Owner creates equivalence statements from his model and metamodel elements to the reference concepts. In the second phase, these equivalence statements are evaluated automatically in order to match model elements and identify conflicts between them. The Model Owner then decides on conflict resolutions and selects the model elements he wants to import from other models. In contrast to classic model merge techniques which result in a consistent union of the integrated models, the Oida framework leaves model specific naming convention and decomposition strategies intact and allows the Model Owners to manage the degree of semantic overlap and thus to keep the complexity of their models down to a practical minimum.

The Oida framework has been designed to facilitate this process. In particular, the design realizes the interactive matching of discipline-specific models with a reference

ontology by employing a BLACKBOARD pattern, which allows a combination of both automated and manual match finder, review, and validation capabilities. The prototypical implementation of the OIDA framework is based on common modeling and ontology frameworks and was evaluated by automated quasi-experiments and interactive case studies with the same sample models as are used in the analysis of conceptual aircraft models, engaging their respective Model Owners. Preliminary unit tests revealed that some of the envisioned automated `KnowledgeSources` could not contribute to the matching process for the given sample models. In particular, the ontology consistency check by the reasoner took considerably more time than should have been required. Therefore, the prototype employs only manual `KnowledgeSources` without implementing the BACKBOARD pattern. Nevertheless, the prototype could identify conflicts between equivalent objects and was able to move selected model elements between given sample models. After participating in the case studies, all Model Owners rated the prototypical Model Matching and Partial Model Merge process better than sufficiently efficient and considered the results plausible.

The analysis of the evaluation suggests three steps for improving the OIDA framework. First, automated `KnowledgeSources` need to be integrated into the OIDA implementation. For instance, the `InferenceModel` can contribute by automated classification of `OntologyResources` and by checking the consistency of ontologies. Thereby, the integration of computationally intensive `KnowledgeSources` to a responsive interactive BLACKBOARD will be included in future research. Secondly, simple equivalence mappings limit the integration capability to one-to-one matchings of concepts. One-to-many mappings would enable the integration of more heterogeneous models and enable the designers to better adapt their tools to their respective methodology. Thirdly, the reference ontology must cover at least all domain concepts of the models involved in an integration process. Additionally, the scope of the reference ontology should be extended to complex data types such as vectors and to behavioral aspects of aircraft systems.

Evolution of the Integration System The case studies showed that at least the initial effort required from the Model Owners in performing the ontology-based model integration process is considerable. However, the extent and the complexity of differences between concurrently evolving models depends on how frequently the integration process is performed. Therefore, how the efforts of the users for subsequent integration processes of concurrently evolving models could be reduced was investigated. The problem was tackled by defining four distinct use cases in which relevant artifacts evolve. Each case was analyzed as to whether the OIDA framework can control this evolution by exploiting available knowledge or applying heuristics in order to limit the effort required from Model Owners and the Ontology Owner. The OIDA design was extended by algorithmic and heuristic `KnowledgeSources` which were implemented in the OIDA prototype. Automated quasi-experiments demonstrated that the OIDA prototype is capable of considerably diminishing the effort required from Model Owners and the Ontology Owner in all four cases of evolution.

8.2. Integrating other Dimensions of Conceptual Models

The OIDA framework demonstrates a solution for collaboration in conceptual aircraft design made possible by the *ex post* introduction of formal semantics into existing models by equivalence statements between model and metamodel elements to shared reference ontology concepts. In principle, designers could assign formal semantics to every model element at the beginning of its information life cycle and modify it if necessary. Thereby, the Match Models and Co-Evolve Integration Artifacts use cases would be reduced to small tasks which could be seamlessly integrated into the modeling work flow. Presumably, the guaranteed availability of object semantics facilitates the extension of ontology-based model integration between other design dimensions such as levels of detail, design variants, or design phases.

The current concept described in this dissertation assumes that a model does not only contain detailed system attributes but also aggregated system attributes which are coupled with the detailed attributes. For instance, the sample models specify the mass of subcomponents such as wing and fuselage, but also explicitly define the aggregated mass of the overall aircraft. This redundancy can be used to check the consistency of algorithms in a conceptual aircraft model. Furthermore, it allows for correlating different models with different levels of detail, if the models is composed the same way. Thereby, it will be a challenge to implement such a reference ontology in a way that it provides a standard composition without limiting the concept designer's flexibility regarding the system topology.

Especially during conceptual aircraft design, not only off-design operation scenarios but also design alternatives are explored and assessed. The latter represent products which significantly diverge by system attributes and also system topology. An early mapping of design alternatives to a common reference ontology could help to identify common model elements and keep them consistent.

Not only the integration of concurrent design models but also the explicit coupling and consistency between models of different design phases is desirable for a sustainable product design process, especially, after the conceptual design phase model elements basically transit to another level of abstraction. For instance, in an early stage of the development a `NewPlaneStarboardEngine` is perceived as an instance object whereas, in later design phases, it is perceived more as a class. For such a class, an appropriate instance object would be `NewPlane001StarboardEngine`. The ultimate challenge for a system integrating different levels of abstraction is the transition between virtual models and physical systems.

A. Implementation of the Oida Prototype

The evaluation of the Oida capabilities described in Chapter 7 was based on an implementation of the essential features of Oida components. Beside the design goals stated in Section 6.1, the following two goals were added for the implementation:

Evaluation Enabler The implementation should facilitate all quasi-experiments and case studies stated in Table 7.1. The objective of all empirical studies is to evaluate the concept of ontology-based user-engaged model integration in the context of conceptual aircraft design, not the verification of the Oida design features. Therefore, capabilities or architectural features of the Oida framework should only be realized if they contribute to the evaluation.

Metamodel Independent The implementation of the Oida should not depend on the metamodel of any specific tool. That is, the implementation should be less dependent on a particular discipline and maintain a general applicability in the context of conceptual aircraft design.

The following section describes the selection of components used for the prototypical implementation in detail.

A.1. Selection of Basic Frameworks

For the development of the Oida framework, the basic design decision was to choose MOF as meta-metamodel and OWL as ontology language. Both are currently the *de facto* standard in their domain.

For the modeling framework, the Eclipse Modeling Framework (EMF) and the Eclipse UML2 framework were considered. EMF is basically an implementation of the essential parts of the MOF specification (eMOF). Furthermore, EMF is the basis for many modeling related frameworks, for example ATL [Jou+08]. The Eclipse UML2 framework is an implementation of the OMG UML 2 specification [OMG11], thus providing more modeling languages constructs than eMOF. However, UML is not commonly used in aircraft modeling. Therefore, specific UML language constructs provided by Eclipse UML2 which are not provided by EMF could not be used. For instance, none of the sample models uses specific language features of UML which are not covered by eMOF. EMF, in turn, provides the capability to dynamically create models and metamodels, which is required for the `Transformation` component described in Section 6.4. Although

ECLIPSE UML2 is based on EMF, this feature is not implemented for UML constructs. Therefore, EMF was selected.

Aircraft Modeling Application

As application platform for OIDA, the PROTÉGÉ ontology engineering tool and OPENCDT, which is based on EMF and the Eclipse Rich Client Platform (RCP), were considered. On the one hand, PROTÉGÉ is focused on creating, visualizing, and querying ontologies as well as performing automated reasoning. Neither the user interface nor the underlying PROTÉGÉ framework provides capabilities required for conceptual aircraft modeling. Previous versions of PROTEGÉ provided import/export services for UML models. However, this plug-in is no longer maintained. As the original sample models are not provided in a UML format, even the previous version of PROTÉGÉ would not be able to access them. On the other hand, OPENCDT can import model files from the tools used to create the sample models. As EMF is profoundly integrated into this platform, not only the model editing capability but also the modeling user interface framework provides a familiar look and feel to the aircraft designer. In contrast to PROTÉGÉ, ontology editing capabilities are not included in OPENCDT. Aiming at the design goal to create the OIDA framework as an extension to existing modeling applications, OPENCDT was chosen as `Aircraft Modeling Application`. However, in order to avoid dependencies on OPENCDT-specific metamodels, the OIDA framework only depends on the ECORE metamodel provided by EMF.

Ontology Framework

For the ontology framework the JENA framework and the PROTÉGÉ API were considered. Both are OWL frameworks implemented in Java and released under an open source license. Whereas the PROTÉGÉ API is made for the PROTÉGÉ Ontology engineering tools, Jena has not obviously been developed for a specific application. Due to its lack of modeling and simulation capabilities, PROTÉGÉ was not an option as an `Aircraft Modeling Application`, Therefore, JENA was chosen as the basic framework for ontology-related capabilities of OIDA. PROTÉGÉ, however, was used during the implementation of OIDA as an ontology engineering tool not only for preliminary experiments and validation of OIDA-generated ontologies but also for the initial implementation of the reference ontology.

These decisions had some implications regarding the choice of `Platform` layer components, the OIDA programming language, the component model, and the concrete syntax of OIDA-generated files. OPENCDT as the `Aircraft Modeling Client` implied the decision for the EMF Client Platform (ECP) [Ecl12a] as `Modeling Client Platform` which provides an EMF specific user interface to models. In particular, aiming for user acceptance, the `ModelNavigator` of ECP was reused to provide a familiar user interface. As a primary programming language, the OIDA framework adopts JAVA from its base frameworks EMF and JENA. Furthermore, the OIDA components are implemented as

	Modeling	Ontology
Specific framework	EMF (eMOF)	Jena (OWL)
Concrete syntax	XMI	RDF-XML
File data format	XML	
Aircraft Modeling Client	OpenCDT	
Modeling Client Platform	ECP	

Table A.1.: Selection of off-the-shelf components for the `Platform`

OSGi plug-ins [OSG12]. Thereby, Oida applications can be loaded as a client extension to the `AircraftModelingApplication`. By facilitating an extension to an existing modeling framework, the second design goal is realized. Regarding the concrete syntax of serialized artifacts, EMF uses XMI which allows XML-based file exchange with other modeling tools. Jena supports several concrete OWL syntaxes. The OWL2 specification [W3C09] states the RDF-XML syntax ensuring compatibility with most other ontology engineering tools. Therefore, fulfilling the third design goal, both models and ontologies are persisted in XML files which is a standard file format.

A.2. ModelProvider Plug-in

The `ModelProvider` implementation is not evaluated directly in this dissertation. However, all evaluated plug-ins require its services. Especially, the `Transformation` plug-in requires an implementation of `ModelElementFilters`. Furthermore, the evaluation of the `OntologyGenerator` required an implementation of `ModelMetrics`. Another practical issue which was raised during the implementation was the capability to determine the `name` attribute of a `Class`, reflectively.

The `Transformation` plug-in requires lists of model and metamodel elements filtered by their MOF-defined properties, e.g., all instance Objects or all directly or indirectly instantiated Classes in a given model. The Oida implementation uses EMF capabilities to extract all model and metamodel elements such as `EObjects`, `Classes`, `EAttributes`, and `EReferences` of a given EMF model. For the implementation of the `ModelElementFilters`, two alternatives were considered: first, an implementation according to the design in Figure 6.3, and secondly, a simple implementation in one class. In accordance with the first implementation goal to enable the evaluation, all `ModelElementFilters` were realized as simple methods of the `Extractor` class.

Similar to `ModelElementFilter`, the architecture for `ModelMetric` was not considered relevant for this evaluation. Therefore, it was decided to provide the capability as methods of the `ModelMetrics` Helper Class. The implementation of particular metrics is described in Section 7.2.1.

For the matching of `disciplineConcepts` with the reference ontology, OIDA must generate a meaningful object identifier. Metamodel Designers usually specify a `name` attribute for this purpose. However, according to MOF the `name` attribute is not mandatory for `Objects`. The `ModelProvider` implements the service to determine the `name` using the reflection capabilities of EMF. Thereby, assigning the name attribute by user interface as part of the Match Models use case or implementing a heuristic was considered. The name attribute determination service is a prerequisite for the `OntologyGenerator`. Therefore, the service must be available before the `Matching` initiates the Match Models use case. Therefore, the name attribute determination service was implemented as heuristical method in `ModelProvider`. The heuristic is analogous to the method used by the `EMF.Edit` generator [Ste+08]. In a first cycle, the heuristic looks for an attribute called `name` using the reflection capabilities of EMF. If unsuccessful, a second search looking for an attribute whose name includes `name` is performed. If still unsuccessful, the JAVA object ID is returned. For the given sample models, this heuristic always returned an accurate name attribute in the first cycle.

A.3. OntologyProvider Plug-in

The `OntologyProvider` plug-in serves as an adapter to the `OntologyFramework`, thus providing an ontology repository to higher layer plug-ins. In particular, the `Transformation` plug-in requires an adapter to an `InferenceModel` implementation. According to the design in Figure 6.4 the `OntologyProvider` also provides implementations of the `OntologyFilter` and `OntologyMetric` interfaces.

The implementation of the `OntologyFrameworkAdapter` classes could be made with only minor adaptions as JENA directly provides most of the required classes and methods. To give an example of an adaption, unlike EMF, JENA methods return multiple objects as iterators instead of lists.

The implementation of the `InferenceModelAdapter` was carried out and tested with the JENA standard reasoner on small artificial models. The design of the OIDA framework and, in particular, the `SemanticValidator` of the `Matching` component is based on the assumption that an automated `Reasoner` requires less than 3 seconds to determine ontology consistency and to produce a meaningful report if an inconsistency is detected. However, a complete classification cycle with more realistic ontologies generated from the sample models required considerably more than 3 seconds. The evaluation strategy is focused rather on the feasibility of an interactive Match Models use case than on the validation of particular automated `KnowledgeSources`. Therefore, it was decided to use PROTÉGÉ via file exchange as an external tool for ontology validation and maintenance. Thereby, the automated integrated reasoning capability was traded with an external application which allows validation of OIDA ontologies offline, testing a variety of reasoners like PELLET [Sir+07] or HERMIT [SMH08]. The file exchange was facilitated by the implementation of the `OntologyFileHandler` adapter which was realized as a HelperClass. Similar to the `ModelProvider` plug-in, a simplified implementation of `OntologyMetrics` was chosen, which is described in more detail in Section 7.2.1.

	Modeling		Ontology TS
	UML	**EMF**	**OWL**
Entities	Class	EClass	Class
	Instance	EObject	Individual
Characteristic	Attribute	EAttribute	Datatype property
Links	Association	EReference	Object property
	Composition	EContainer	N/A (upper ontology import)
	Superclass	ESuperType	Superclass

Table A.2.: Mappings between modeling and ontology concepts. For the modeling context both the UML and the corresponding concepts in EMF are listed. As Composition is not specified by OWL, the concept has to be imported from the `mereology.owl` upper ontology which is a part of the `referenceOntology`.

A.4. Transformation Plug-in

The `Transformation` subsystem, which a critical component of the OIDA framework, is evaluated separately in Section 7.2.1. In order to provide the ontology-based model transformation service, `OntologyGenerator`, MetamodelGenerator, and `ModelTransformation` were essentially implemented according to Figure 6.6.

The `OntologyGenerator` uses the `ModelElementFilters` for `InstanceObjects`, `Classes`, and `StructuralFeatures` of the `ModelProvider` to generate an OWL ontology according to the model $\leftrightarrow$ *ontology* mapping schema in Table A.2 and a `RenamerStrategy`. The concept mapping schema, which is in accordance with the guidelines published by the Object Management Group [Obj09], has no direct mapping between Composition in UML or Containment in EMF and OWL. It has to be expressed by an upper ontology which is designed according to Rector et al. [Rec+05]. Appendix B describes this mereology upper ontology in more detail. The transformation realized by the `OntologyGenerator` is a homomorphism, as every `Object` is transformed into an `Individual`. However, it is not an isomorphism as not all possible concepts of an ontology can be transformed into a model. For instance, OWL constructs, such as Antisymmetric Property or Anonymous Class, cannot be translated to an EMF model.

The data types specific to modeling tools for attribute values are not only used to increase the model integrity but also to express semantics. Mapping data types between models and ontologies preserves not only information which can be evaluated for semantic matching but also allows the reuse of native methods for comparing and converting attribute values. Whereas the conversion of primitive data types is already implemented in the JENA framework, it requires the developer to register model-specific data types in a type registry. Up to now, this capability has been evaluated with the `Unit` data type used in the APD and SIMCAD sample models which implements SI and non-SI units

of measures. OWL does not offer special data types but uses the data types defined in the XML Schema Definition (XSD). The first prototype of the `OntologyGenerator` translated domain-specific values of data types not covered by the JENA framework to a `String` representation. However, this implementation did not allow adoption of discipline-specific data types to the `referenceOntology`. Furthermore, it hindered the reuse of implementations of type-specific methods determining equality. For instance, the `Unit` class used in the APD and SIMCAD sample metamodels allows conversion between different units of measure. Therefore, the `Transformations` between models and ontologies such as the `OntologyGenerator` and the `MetamodelGenerator` implement an adapter to the capabilities of the JENA framework. JENA provides a bidirectional type mapping registry which allows registering user-defined data types by *ad hoc* creation of non-standard data types in an ontology .

One of the core features of the OIDA framework is the matching of objects which represent the same physical object. This capability is realized by the `OntologyGenerator` transforming EMF models to OWL ontologies. However, both EMF and OWL have different concepts of object identity. Basically, an OWL ontology is a set of statements about `OntologyResources` and their relations to each other. Thereby, every `OntologyResource` has a Unified Resource Identifier (URI). According to the design requirement for an injective model-to-model mapping, a renaming schema must generate a unique URI for every Object. The URI should be readable for humans to allow low level ontology maintenance and plausibility checks. In that sense a URI such as `http://bhl.net/aircraft.owl#wingspan` is preferable to `http://bhl.net/aircraft.owl#105154`. Therefore, the implementation of a suitable `RenamerStrategy` was based on three assumptions. First, that almost every object has a name attribute; secondly, that the value of the `name` attribute of a model element indicates its meaning; and thirdly, that the value of the `name` attribute of its container object indicates the context. EMF stipulates that every metamodel element like `EClass`, `EAttribute`, and `EReference` has a `name` attribute which informally indicates the semantics of the metamodel element comprehensible for the developer. Instances of `EObject`, which is the root class of `ECore`, do not have a specified name attribute. Therefore, `RenamerStrategy` implementations use the `name` attribute heuristic of the `ModelProvider`. Furthermore, in EMF every `EObject` provides an `EContainer` reference which allows navigation of the containment tree of a model. Taking Object Identity, `name` attribute value, and `EContainer` references as sources, three alternatives for a `RenamerStrategy` were considered during implementation:

ClassName:ObjectName Whereas in EMF the name of a class is unambiguous in a certain name space, its combination with the object name is ambiguous in EMF, as the object name is an arbitrary user-defined attribute which in a valid model can be used more than once. Furthermore, this renaming schema abandons the context within the model and provides no version information about the object.

Object Identity The Object Identity is unambiguous within a runtime environment (see Section 2.1) However, the Object Identity deliberately abandons the conceptual and structural context. Accordingly, it does not represent the semantic identity of an object. Furthermore, an Object Identity which is unreadable for humans makes interactive low level ontology maintenance and engineering impossible.

ContainerName.ObjectName This renaming schema includes the context of the object. Furthermore, after a first manual inspection of the sample models, it was considered to be unambiguous for the given models. Therefore, this `RenamerStrategy` was implemented in the OIDA framework.

The decision for the selection of a `RenamerStrategy` was based on the necessity of the development process to inspect the ontologies in order to verify the algorithms. Therefore, the `ContainerName.ObjectName` strategy was considered the most adept alternative. Consequently, the implementation of this `RenamerStrategy` was used in the evaluation described in Section 7.2.1, Section 7.2.2, and Section 7.2.3. The evaluation of the transformation T_{MO} described in Section 7.2.1, however, revealed that the first container level is sufficient to rename model `EAttributes` and `EReferences` unambiguously but not to rename `EObjects`.

The `MetamodelGenerator` creates the `referenceMetamodel` from the `referenceOntology` using the mapping schema shown in Table A.2 and the type mapping capability as used by the `OntologyGenerator`, but in the opposite direction. The implementation not only employs `OntologyFilters` to iterate through different kinds of reference concepts but also uses the capability of the EMF framework to dynamically generate a metamodel. In contrast to the `OntologyGenerator`, `MetamodelGenerator` uses the local name suffix of the URI for naming metamodel elements, as the URI is unique by definition.

The `ModelTransformation` in Figure 6.6 transforms a `disciplineModel` with semantically overlapping content to a common metamodel. Performing this transformation two `disciplineModels`, conforming to different metamodels, can be compared and merged using existing metamodel-based frameworks. The implementation of `ModelTransformation` is based on the assumption that the direct model-to-model transformation can be determined indirectly by the equivalence statements between instance and metaconcepts of `disciplineOntology`, derived from the `disciplineModel` to the `referenceOntology`.

This transformation is a homomorphism regarding the model representation of `EObjects`, `EClasses`, and `EAttributes`. The OIDA framework disregards behavior and links of entities to other entities. Accordingly, the OIDA framework is neither able to detect conflicts of these model element features nor does it support their migration from one model to the other. During th runtime of the `Transformation` plug-in, the `ModelTransformation` stores the mapping between each `disciplineObject` and `referenceObject` in a mapping model. According to its design, the `ObjectMapping` also contains a map-

ping between disciplineClass and referenceClass and the respective Attributes. Thereby, every discipline-specific object can be mapped to a reference classifier, independent of the mapping its discipline-specific classifier as long as ontology consistency is maintained. The ObjectMapping data model has been specified as an ECORE model and generated using the EMF code generator.

The capability of the Transformation plug-in to map between data types which are used by the OntologyGenerator and MetamodelGenerator, is basically an adapter to existing JENA features. The implementation of the ModelTransformation extends this capability by ontology-based type conversion which is required to merge attribute values from different but related data types. For instance, in the SIMCAD sample model, the attribute representing the number of passengers is a Double type variable, whereas in APA the same concept is an Integer type variable. JAVA supports implicit type casting from Integer to Double. The reverse operation requires the Model Owner's declaration that the number of passengers is always an Integer value disregarding the range of the concrete data type in the disciplineModels. If the attribute transformation algorithm does not find the data type of the source attribute in the target attribute, it queries the referenceOntology for another type representing a larger number set. This would allow implicit type casting. If this is not successful, it queries the referenceOntology for a data type representing the next smaller number set which would allow number conversion. If this is still not successful, the target attribute value is not set. However, the type conversion method only implements the conversions required for the sample models and emits a warning of potential data loss if implicit type casting is not applicable.

The mapping between the source and the target of ModelTransformation can be deleted after the Merge Models Partially use case as they are not required during the runtime of the Transformation plug-in. Thereby, the amount of redundant coupled artifacts is reduced.

A.5. Matching Plug-in

The design of the Matching features a BLACKBOARD pattern which is based on the assumption that simple automated heuristics and algorithms can contribute to partial solutions which are aggregated and exchanged via the Blackboard and orchestrated by the BlackboardControl. The design stipulates three KnowledgeSources: NameEquivalenceFinder, StructuralEquivalenceFinder, and SemanticValidator. Due to the previous experience with the implementation of the InferenceModel, it was decided to first develop and evaluate each potential KnowledgeSource for the sample models, before tackling the implementation of the BLACKBOARD pattern.

The design NameEquivalenceFinder is based on the assumption that if objects have the same name they most probably represent the same thing. The capability was developed and tested with artificial aircraft model data. However, tests with sample data revealed no directly matching names between the referenceOntology and the disciplineOntologies. Therefore, this knowledge source was considered ineffective for the evaluation.

The StructureEquivalenceFinder is based on the assumption that semantically equivalent objects can be identified by their structure. Accordingly, equivalence of the attributes of two objects is a strong indication that the two objects represent the same thing even if the object identifiers are different. The StructureEquivalenceFinder was implemented evaluating equivalence statements between attributes. It creates an EquivalenceHypothesis between two classes if there is a bijective equivalence mapping between the attributes of the classes. This capability was successfully tested with sample models: After manually mapping three attributes, the StructureEquivalenceFinder was able to match a metamodel class.

The SemanticValidator was implemented using the InferenceModel adapter implementation of the ModelProvider plug-in and successfully tested with an artificial model. However, the drawbacks of the off-the-shelf reasoner described in Appendix A.3 applied. The unit test required over 20 seconds for a complete classification of a simplistic test model.

With NameEquivalenceFinder and SemanticValidator being ineffective for the given sample models, it was unlikely that the StructuralEquivalenceFinder alone would improve the efficiency of the Match Models use case significantly. Therefore, it was decided to implement only the EquivalenceHypothesis commands necessary for manual generation and validation of matching. Thereby, the EquivalenceHypothesis data model was specified as an ECore model and generated using the EMF code generators.

A.6. Merging Plug-in

The Merging plug-in compares two disciplineModels and determines non-conflicting SemanticMatches, ConflictCandidates, and ImportCandidates. As shown in Figure 6.9, the Merging is based on ModelTransformation which is provided by the Transformation plug-in implementation. Moreover, the EMF Compare framework [Ecl12b] was used as a template for the data model and for validating the implementation. Accordingly, the data model, which was specified as an ECore model, stipulates a DiffModel which contains all instances of SemanticMatch and its subclasses.

In the Merge Models Partially use case, the Model Owner reviews the identified SemanticMatches regarding their plausibility. Therefore, the user interface facilitating Match Models requires a service which displays the attributes of the matched objects without exposing the user to the naming convention and data types of the source model. This service was implemented by IntegrateAttribute which can handle missing attributes and unset values. Furthermore, the interaction design of the Merge Models Partially use case requires the capability to measure the state of the DiffModel for progress report. The MergingMetrics implements this service by analyzing the DiffModel with respect to reported and confirmed conflicts and reported or selected import candidates. The MergingControl implementation initiates the TransformationControls and the generation of the DiffModel, and propagates conflict resolution and import decisions to the targetDisciplineModel and the targetDisciplineOntology.

A.7. Evolution Plug-in

The `Evolution` plug-in facilitates the Co-Evolution capability of the OIDA framework. According to the design depicted in Figure 6.10, this capability is accomplished by two kinds of `KnowledgeSources`: `EvolutionAlgorithm` and `EvolutionHeuristic`. However, due to the ineffectiveness of most automated `KnowledgeSources` the BLACKBOARD pattern was not implemented. Therefore, `EvolutionAlgorithm` and `EvolutionHeuristic` have been implemented as automated preprocessors to the manual Match Models use case. These classes are described in detail in Section 7.2.4.

A.8. MatchingApp Plug-in

The purpose of the `MatchingApp` plug-in is to provide a user interface for the Match Models process. According to the design in Figure 6.13, the plug-in implements the `ManualEquivalenceCreator` and the `EquivalenceReviewForm` as `KnowledgeSource`. As the BLACKBOARD pattern was implemented, these classes are directly controlled by the `MatchingAppControl`. The `MatchingApp` plug-in is started by the implementation of an ECP `ModelElementOpener`. When the `MatchingApp` plug-in is loaded to the `AircraftModelingApplication`, the underlying ECP framework lets the Model Owner select the `disciplineSpecificRootObject` in his model and start the `MatchingApp` via a context menu.

The `ExtendingMatchingEditor` integrates the `ManualEquivalenceCreator` and the `EquivalenceReviewForm`. During the initialization of the Editor the `ImportPreviousMappingsDialog` is shown, which allows the user to load an OWL file which contains equivalence statements from a previous Match Models session. This dialog controls an `ImportPreviousMappings` instance of the `Evolution` plug-in.

When the import is complete, the user interface depicted in Figure A.1 is displayed. This Editor is implemented reusing the JFACE and the SWT framework provided by ECP. Thereby, the Model Owner experiences a look-and-feel he is familiar with from the `Aircraft Modeling Platform`. If the Model Owner does not find the appropriate `referenceOntologyResource` during the Match Models use case, he can select the `disciplineOntologyResource` and click the "Add Reference" button to open a `CreateReferenceResourceDialog` where the appropriate name for the new `referenceOntologyResource` can be entered. When the user confirms the dialog, the defined resource is created in the `referenceOntology` and an equivalence statement is added to the `disciplineOntologyResource`. Upon returning to the `ExtendingMatchingEditor`, the user cannot only save the mapped `disciplineOntology` but also the extended `referenceOntology` to an OWL XML file.

The `Extending Matching Editor` serves as a demonstrator for a user interface which integrates the OIDA framework capabilities which supports the Match Models and Co-Evolve Integration Artifacts use cases. For the evaluation of the Match Models use case a special implementation of the `MatchingEditor` was used which does not allow the

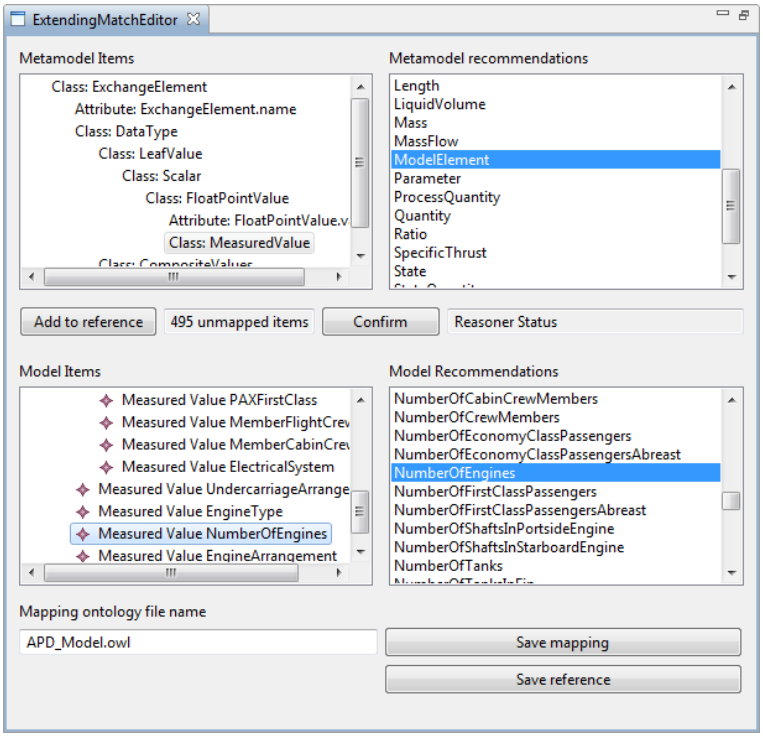

Figure A.1.: A screen shot of the `ExtendingMatchingEditor`

extension of the `referenceOntology`. This reduced user interface is described in detail in Section 7.2.2.

A.9. MergingApp Plug-in

The `MergingApp` provides a user interface which enables the Model Owner to review `SemanticMatchings` and decide on `ConflictCandidates` and `ImportCandidates`. The `MergingApp` was implemented as shown in Figure 6.14. In particular, the `SemanticMatchReviewForm` and the `ImportCandidateReviewForm` are implemented as SWT `Dialogs`.

Like `MatchingApp`, the `MergingApp` plug-in is started by the implementation of an ECP `ModelElementOpener` interface. Consequently, the Model Owner can select the `disciplineTargetRootObject` in his model and start the `MergingApp` via a context menu. As the Merge Models Partially stipulates a predefined `disciplineSourceModel`, a selection of a `disciplineSourceRootObject` is not necessary for the evaluation. Accordingly, the appropriate model and ontology data are loaded automatically. Furthermore, the mapped `disciplineOntologies` are loaded and the `MatchingControl` is initiated. When `MatchingControl` has finished creating the `DiffModel`, the `SemanticMatchReviewForm` for matches and conflicts and the `ImportCandidateReviewForm` are displayed consecutively. Both `Dialogs` are implemented based on the JFace

and SWT framework. In particular, the `Dialogs` use the `IntegratedAttribute` class of the `Merging` plug-in to provide a consistent view of the incoming model data to support the user in assessing the plausibility of the matching results. The `SemanticMatchReviewForm` displays the detected `SemanticMatches` and `ConflictCandidates` and receives the user's confirmation or report on implausible matches. Thereby, confirmation of a `ConflictCandidate` is interpreted as a user decision to resolve the conflict. Accordingly, when the user pushes the button labeled "Next" to conclude the Dialog, the `MergingControl` resolves the conflict immediately and the `ImportCandidateReviewForm` instance is opened.

MigrateDialog

The `ImportCandidateReviewForm` is the user interface to `MergingControl` which facilitates the interactive process of the Migrate Model Parts use case. The `Dialog` depicted in Figure A.3 displays `ImportCandidates`. The Model Owner can either discard a proposed `ImportCandidate` or select it for import. In the first case, he can give an additional review comment. In the latter case, he can define an appropriate `disciplineSpecificName` and `disciplineSpecificContainer` object for the respective model element selected for import in order to integrate the elements in the naming convention and decomposition strategy specific to the target model. The possible containers are shown in a `ContainerNavigatorForm` which is integrated on the left side of the `ImportCandidates` dialog. The UI also provides the current attribute values of the selected model element, reusing `IntegrateAttribute`. The user concludes the dialog by pushing the finish button and thereby triggers `MergingControl` to propagate the confirmed `ImportCandidates` to the `disciplineTargetModel` and the `disciplineTargetOntology`.

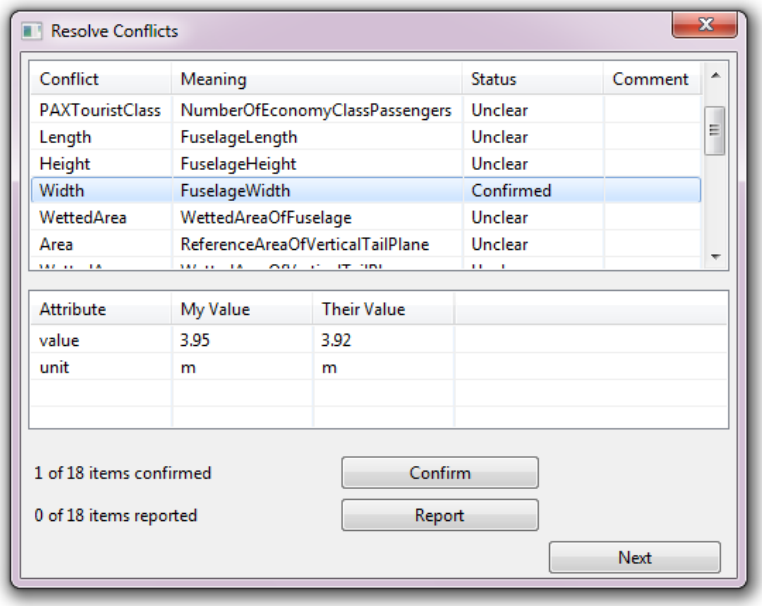

Figure A.2.: A screen shot of the `ConflictResolutionDialog` during the resolution of an attribute conflict between a SIMCAD Chief Engineering Model element and an APD model element representing the width of an aircraft fuselage

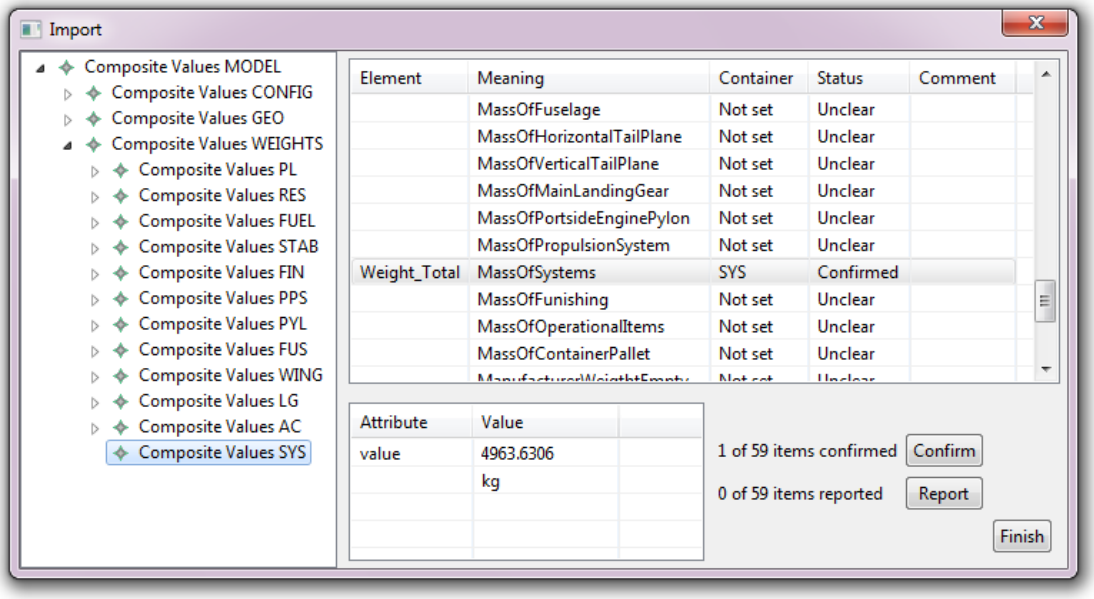

Figure A.3.: A screen shot of the `MigrateDialog` during the import of model elements from the SIMCAD Chief Engineering Model to the APA target model

B. Implementation of the Reference Ontology Prototype

The following chapter describes the development of the `referenceOntology` as it was used in the evaluation of Match Models and Merge Models Partially, and its revision during the evaluation of the Co-Evolve Integration Artifacts.

The design of the reference ontology is driven by the ontology mapping capabilities of the OIDA framework and the need to agree on a shared conceptualization between the Model Owners. If the OIDA only supports a simple one-to-one equivalence mapping between concepts, the reference ontology must be structurally similar to the discipline-specific models. In addition, the reference ontology must exhibit a structure and naming convention familiar to the Model Owners in order to facilitate not only the review of existing conceptualizations but also the agreement on additional domain concepts. Accordingly, the reference ontology has the following design goals:

Clear scope The Model Owners must agree upon the semantics of the concepts in the reference ontology. A clear and narrow scope facilitates the agreement, e.g. basic concepts of an aircraft like wingspan, fuselage and number of passengers. In the course of model matching sessions, more concepts from the models are added to the reference ontology by the respective Model Owners. Thereby, the Ontology Owner ensures that the reference ontology does not contain concepts specific to one particular methodology or tool. Generally, tool specific concepts are not useful for semantic model matching, as they are unlikely to appear in other models and will therefore be difficult to agree upon.

Clear minimum content In order to serve as an effective basis for model integration, the reference ontology contains all concepts from the discipline-specific models which essentially describe the subject matter. In a conceptual design process a Chief Engineering Model is used to coordinate the design activities. Generally, this model has no specific focus, a limited level of detail, and can be used as a template for the minimum content. For example, among the sample models the SIMCAD sample model has the lowest level of detail in most aspects of the aircraft. Therefore, the baseline version of the reference ontology for the evaluation in Chapter 7 was derived from the SIMCAD sample model. Furthermore, the reference ontology should contain conceptualizations of shared metamodel elements. These reference metaconcepts are essential for matching models conformal to similar metamodels.

Focus on knowledge representation Stemming from the different purpose and intent, ontologies follow design patterns [GP09] other than models. However, the OIDA frame-

work supports model owners in expanding the reference ontology by transferring entities and structures from source models to the reference ontology. The Ontology Owner is responsible for avoiding discipline-specific modeling design patterns in the reference ontology that are less suitable for knowledge representation and automated reasoning.

Based on standard upper ontologies The reference ontology should import fundamental concepts of the application domain from standard upper ontologies. For instance, the domain of conceptual aircraft design is based on fundamental concepts like physical quantities and units of measure which can be imported into the reference ontology from existing upper ontologies, such as the "Library for Quantity Kinds and Units" [Kon+11]. This practice of modularization promotes the compatibility of the reference ontology to other domain ontology creation efforts.

The implementation of the `referenceOntology` was oriented towards the implementation goals of OIDA aiming at enabling the evaluation and maintaining independence from specific conceptual design tools. The trade-off between the primary purpose and the design goals raised three issues regarding the scope of the `referenceOntology`, the minimum complexity, and which upper ontology to import.

As mentioned above in Section 1.2.4, the scope of the model integration process described in this dissertation has to do with the structural aspects of an aircraft. The structural description is very concrete compared to other aspects such as aircraft system behavior. Therefore, the structural aspects are a good basis to expand the `referenceOntology` to other aspects.

The minimum level of detail mentioned in the `referenceOntology` design goals is a measure for the complexity of the `referenceOntology`, i.e. the number of entities and links between the entities. The demonstration of ontology-based model integration required at least the coverage of the entities specified in the metamodels involved and the objects of the master model. Preliminary analysis had shown that the SIMCAD sample model was most adept for serving as a Chief Engineering Model. Therefore, the first version used in the evaluation of Model Matching and Partial Model Merge was simulated by the Ontology Owner with the SIMCAD sample model, starting with an initially empty `referenceOntology`. Subsequently, the `referenceOntology` was extended by the Ontology Owner performing the Match Models use case with the APD sample model. The APA sample model was not included in the preparation process for two reasons. First, the naming convention of the APA sample model was very difficult to comprehend for an Ontology Owner unfamiliar with the APA tool. Without the Model Owner of APA it would have been difficult to determine the meaning of the concepts and to decide whether the particular concept was within the scope of the `referenceOntology`. Secondly, performing the Match Models use case with the APA sample model using a `referenceOntology` which was not prepared by the Ontology Owner allowed investigation of the consequences of such an incomplete reference to the efficiency and user acceptance of the APA Model Owner.

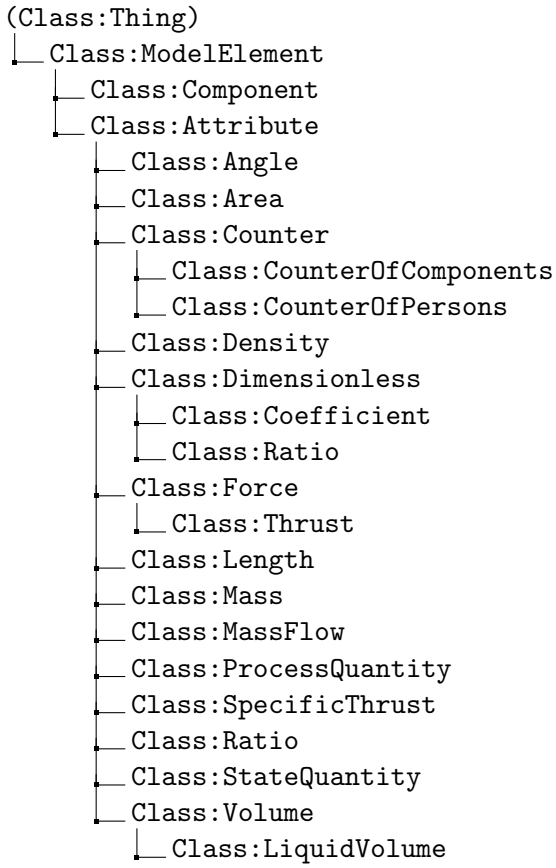

Figure B.1.: The `OntologyClasses` of the `referenceOntology` in a taxonomy representation. Especially the `Parameter` class is differentiated into many subclasses in order to keep the number of individuals for each `OntologyClass` low.

The simulation of the Match Models use case ensured that during the case studies the ontology matching could be established by simple equivalence statements. Consequently, the `referenceOntology` basically adopted the distinction between instances and metaconcepts from the sample models. However, this proceeding aligned the structure of the `referenceOntology` more to the specific conventions of the sample models and thus moved the implementation of the `referenceOntology` away of the goal to create a knowledge representation independent from concrete conceptual models. Furthermore, in contrast to most publicly available upper ontologies, the `referenceOntology` has a small number of `OntologyClasses` and a high number of `Individuals`. As a consequence, founding the `referenceOntology` on established external upper ontologies was not attempted. Figure B.1 shows the `Classes` of the `referenceOntology` in a taxonomy representation. It contains a `NamedClass` representation for every class in the metamodels of the sample models.

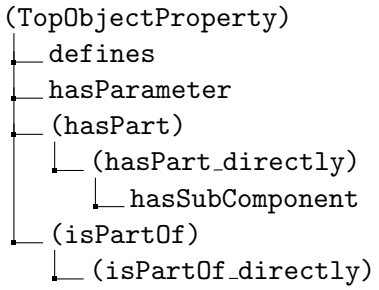

Figure B.2.: The taxonomy of `ObjectProperties` in the `referenceOntology`. The `referenceOntology` uses the same mereology upper ontology as the model to ontology transformation T_{MO} described in Section A.4.

On the other hand, using the Match Models use case for the initial creation of the `referenceOntology` limited the complexity of `referenceOntology` to a necessary minimum required to demonstrate the ontology-based model integration process. Furthermore, Model Owners were more likely to familiarize with a reference similar to their respective model, thus, being more efficient during the mapping process and more willing to accept the model integration process. Additionally, the Match Models use case stipulates a generic naming convention for newly created reference concepts. Therefore, the process conveys a `referenceOntology` less dependent on a specific sample model.

The implementation of the `MatchingApp` only evaluates inheritance relations and `DatatypeProperties` in order to limit the number of matching candidates. Therefore, inheritance relationships were created in the `referenceOntology` aiming to keep the number of direct instances of every `OntologyClass` under five `Individuals`. To express whole-part relationships on the metaconcept level, the `mereology.owl` upper ontology was implemented according to Rector et al. [Rec+05]. Figure B.2 shows the `ObjectProperties` of the `referenceOntology` in a taxonomy representation which include the `mereology.owl`. As for now, no additional upper ontology was imported.

The `DatatypeProperties` depicted on the left side of Figure B.3 were derived from attributes of classes of the sample models. Thereby, the goal was to provide a complete coverage of all attributes, which is necessary especially for `ModelTransformation` which generates the input for `ModelComparator`. These `DatatypeProperties` were used in the case studies described in Section 7.2.2 and Section 7.2.3 that evaluated the Model Matching and Partial Model Merge capabilities. These case studies revealed that the $\mathbb{Z}$ and $\mathbb{R}$ number sets were not adequately represented, which led to substantial ambiguity of the `RenamerStrategy` employed in `OntologyGenerator`. Therefore, the `DatatypeProperties` of the `referenceOntology` were extended by the `hasRealNumberValue` property and its subproperty `hasIntegerNumberValue`. The revised version of the `DatatypeProperty` taxonomy depicted on the right side of Figure B.3 was used in the quasi-experiments evaluating the Co-Evolution capabilities of the OIDA framework, in particular, regarding the type mapping capability described in Section A.4.

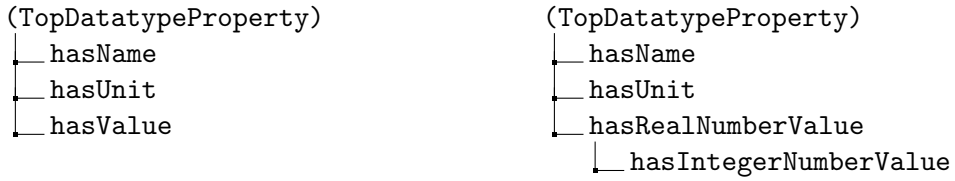

Figure B.3.: The `DatatypeProperties` of the `referenceOntology` in a taxonomy representation. The tree on the left shows the first version in the case studies of the evaluation. The tree on the right shows the second version used in the quasi-experiments evaluating the co-evolution capabilities of the OIDA framework.

In general, all `ObjectProperties` and `DatatypeProperties` in `referenceOntology` have a verb as prefix in order to express a relation between entities.

An excerpt of the reference Individuals embedded in the taxonomy of their respective reference Classes is depicted in Figure B.4. These reference individuals were used throughout the evaluation described in Chapter 7.

Both versions of the `referenceOntology` described can be expressed by the OWL-LITE syntax.

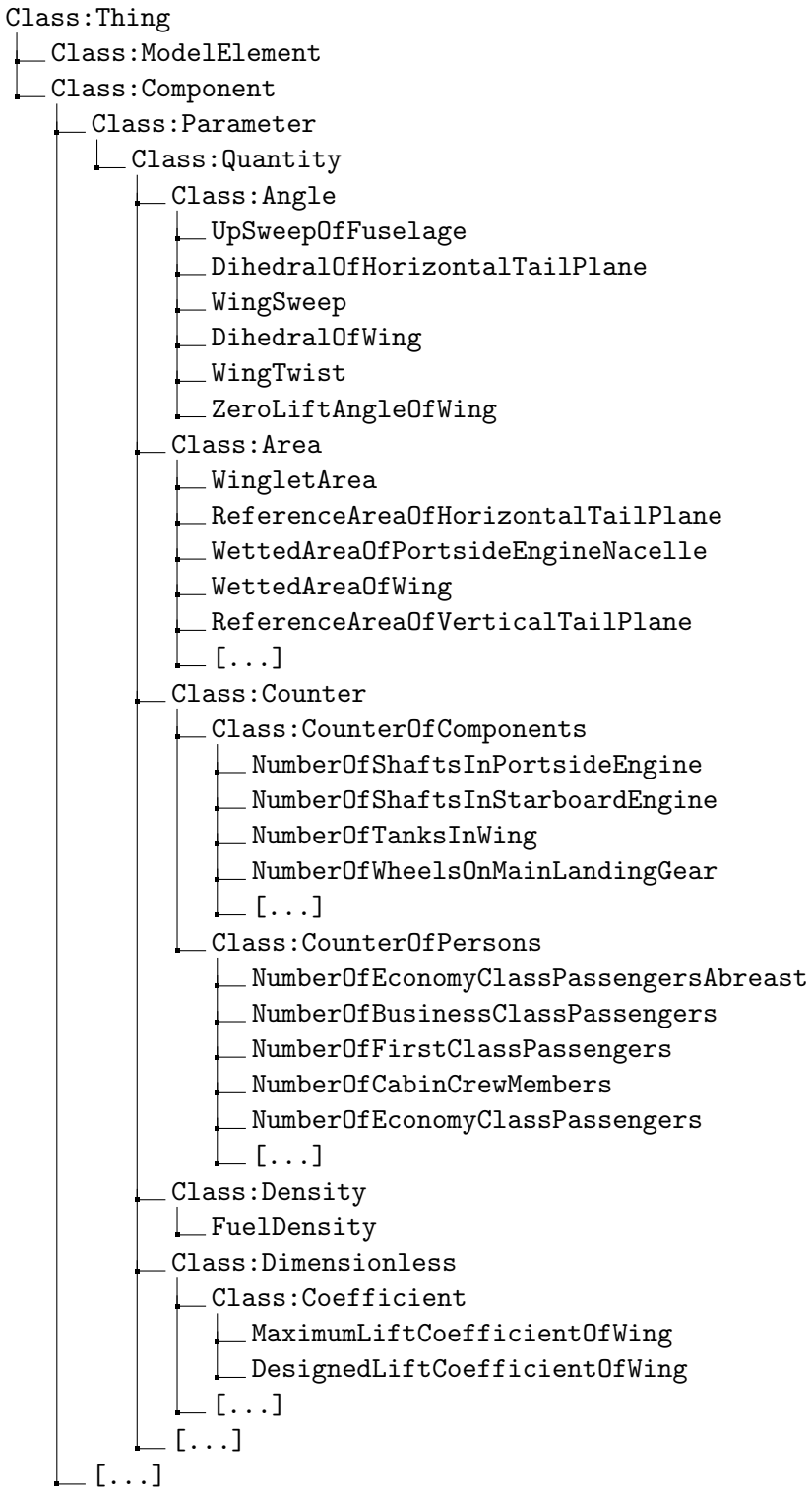

Figure B.4.: An excerpt of the `Individuals` of the `referenceOntology` depicted as instances of the class taxonomy

List of Figures

List of Tables

Bibliography

[Ada84] John G. Adair. "The Hawthorne effect: A reconsideration of the methodological artifact." In: *Journal of Applied Psychology* 69.2 (1984), p. 334.

[AK03] Colin Atkinson and Thomas Kuhne. "Model-Driven Development: A Metamodeling Foundation". In: *Software, IEEE* 20.5 (Sept. 2003), pp. 36 –41.

[BHS07] Frank Buschmann, Kevlin Henney, and Douglas C. Schmidt. *Pattern-Oriented Software Architecture: A Pattern Language for Distributed Computing*. 1st ed. Vol. 4. John Wiley & Sons, 2007.

[BHS08] Franz Baader, Ian Horrocks, and Ulrike Sattler. "Description Logics". In: *Handbook of Knowledge Representation*. Ed. by Frank van Harmelen, Vladimir Lifschitz, and Bruce Porter. Vol. 3. Foundations of Artificial Intelligence. Elsevier, 2008, pp. 135 –179.

[Böh12] Daniel Böhnke. *CPACS – A Common Language for Aircraft Design*. Apr. 2012. URL: `http://software.dlr.de/p/cpacs/home/`.

[Bru+06] Greg Brunet, Marsha Chechik, Steve Easterbrook, Shiva Nejati, Nan Niu, and Mehrdad Sabetzadeh. *A Manifesto for Model Merging*. Shanghai, China, 2006.

[DFV07] Marcos Didonet Del Fabro and Patrick Valduriez. "Semi-automatic Model Integration using Matching Transformations and Weaving Models". In: *Proceedings of the 2007 ACM symposium on Applied computing*. SAC '07. ACM, 2007, pp. 963–970.

[Dil88] Craig Dilworth. "Identity, Equality and Equivalence". In: *Dialectica* 42.2 (1988), pp. 83–92.

[Doa+03] AnHai Doan, Jayant Madhavan, Robin Dhamankar, Pedro Domingos, and Alon Halevy. "Learning to Match Ontologies on the Semantic Web". In: *The VLDB Journal* 12 (4 2003), pp. 303–319.

[Ecl12a] Eclipse Foundation. *EMF Client Platform project home*. 2012. URL: `http://eclipse.org/emfclient/`.

[Ecl12b] Eclipse Foundation. *EMF Compare*. 2012. URL: `http://eclipse.org/emfcompare/`.

[Fel10] Christiane Fellbaum. "WordNet". In: *Theory and Applications of Ontology: Computer Applications*. Ed. by Roberto Poli, Michael Healy, and Achilles Kameas. Springer Netherlands, 2010, pp. 231–243.

[Gam+94] Erich Gamma, Richard Helm, Ralph E. Johnson, and John Vlissides. *Design Patterns*. 1st ed. Addison-Wesley Reading, MA, 1994.

[Gaš+04] Dragan Gašević, Dragan Djurić, Vladan Devedžić, and Violeta Damjanović. "Converting UML to OWL Ontologies". In: *Proceedings of the 13th international World Wide Web conference on Alternate track papers & posters*. WWW Alt. '04. ACM, 2004, pp. 488–489.

[GB99] Erich Gamma and Kent Beck. "JUnit: A Cook's Tour". In: *Java Report* 4.5 (1999), pp. 27–38.

[GP09] Aldo Gangemi and Valentina Presutti. "Ontology Design Patterns". In: *Handbook on Ontologies*. Ed. by Steffen Staab and Rudi Studer. International Handbooks on Information Systems. Springer Berlin Heidelberg, 2009, pp. 221–243.

[Gro11] Object Management Group. *OMG's Meta Object Facility (MOF) – Home Page*. Dec. 2011. URL: `http://www.omg.org/mof/`.

[Gro12] Object Management Group. *OMG SysML*. Apr. 2012. URL: `http://www.omgsysml.org/`.

[Gru09] Tom Gruber. "Ontology". In: *Encyclopedia of Database Systems*. Ed. by Ling Liu and M. Tamer Özsu. First. Springer Publishing Company, Incorporated, 2009.

[HBJ09] Markus Herrmannsdoerfer, Sebastian Benz, and Elmar Juergens. "Automatability of Coupled Evolution of Metamodels and Models in Practice". In: *Model Driven Engineering Languages and Systems*. Ed. by Sophia Drossopoulou. Vol. 5653. Lecture Notes in Computer Science. Springer Berlin/Heidelberg, 2009, pp. 52–76.

[Hen11] B. Henderson-Sellers. "Bridging metamodels and ontologies in software engineering". In: *Journal of Systems and Software* 84.2 (2011), pp. 301–313.

[HH10] Harry Halpin and Patrick. J. Hayes. "When owl:sameAs isn't the Same: An Analysis of Identity Links on the Semantic Web". In: *Proceedings of the WWW2010 workshop on Linked Data on the Web, LDOW2010*. 2010.

[HP08] Jon Holt and Simon Perry. *SysML for Systems Engineering*. Vol. 7. The Institution of Engineering and Technology, 2008.

[HVW11] Markus Herrmannsdoerfer, Sander Vermolen, and Guido Wachsmuth. "An Extensive Catalog of Operators for the Coupled Evolution of Metamodels and Models". In: *Software Language Engineering*. Ed. by Brian Malloy, Steffen Staab, and Mark van den Brand. Vol. 6563. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2011, pp. 163–182.

[ISO94] ISO. *Industrial Automation Systems and Integration – Product Data Representation and Exchange – Part 1: Description Methods: Overview and Fundamental Principles*. Tech. rep. ISO 10303-1. International Organization for Standardization, Geneva, Switzerland., 1994.

[JAS10] Ricardo Jardim-Goncalves, Carlos Agostinho, and Alfonso Steiger-Garcao. "Sustainable Systems' Interoperability: A reference model for seamless networked business". In: *Systems Man and Cybernetics (SMC), 2010 IEEE International Conference on*. Oct. 2010, pp. 1785 –1792.

[JBR99] Ivar Jacobson, Grady Booch, and James Rumbaugh. *The Unified Software Development Process*. Addison-Wesley Object Technology. Pearson Education, Limited, 1999.

[Jou+08] Frédéric Jouault, Freddy Allilaire, Jean Bézivin, and Ivan Kurtev. "ATL: A model transformation tool". In: *Science of Computer Programming* 72.1 – 2 (2008), pp. 31 –39.

[KBA02] Ivan Kurtev, Jean Bézivin, and Mehmet Akşit. "Technological Spaces: An Initial Appraisal". In: *International Conference on Cooperative Information Systems (CoopIS), DOA'2002 Federated Conferences, Industrial Track, Irvine, USA*. Oct. 2002, pp. 1–6.

[KC86] Sertag N. Khoshafian and George P. Copeland. "Object Identity". In: *Conference proceedings on Object-oriented programming systems, languages and applications*. OOPLSA '86. Portland, Oregon, United States: ACM, 1986, pp. 406–416.

[Kög11] Maximilian Kögel. "Operation-based Model Evolution". PhD thesis. Technische Universität München, 2011.

[Kon+11] Hans Peter de Koning, Nicolas Rouquette, Roger Burkhart, Huascar Espinoza, and Laurent Lefort. *Library for Quantity Kinds and Units*. 2011. URL: `http://www.w3.org/2005/Incubator/ssn/ssnx/qu/qu.html`.

[Kra+06] Gerti Kramler, G. Kappel, T. Reiter, E Kapsammer, W. Retschitzegger, and W. Schwinger. "Towards a Semantic Infrastructure Supporting Model-based Tool Integration". In: *Proceedings of the 2006 international workshop on Global integrated model management*. GaMMa '06. ACM, 2006, pp. 43–46.

[Maa10] Walid Maalej. "Intention-Based Integration of Software Engineering Tools". PhD thesis. Technische Universität München, 2010.

[Mel+02] Stephen Mellor, Kendall Scott, Axel Uhl, and Dirk Weise. "Model-Driven Architecture". In: *Advances in Object-Oriented Information Systems*. Ed. by Jean-Michel Bruel and Zohra Bellahsene. Vol. 2426. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2002, pp. 233–239.

[Noy04] Natalya F. Noy. "Semantic Integration: A Survey Of Ontology-Based Approaches". In: *SIGMOD Rec.* 33.4 (2004), pp. 65–70.

[Obj09] Object Management Group. *Ontology Definition Metamodel*. Tech. rep. OMG, 2009. URL: `http://www.omg.org/spec/ODM/1.0/`.

[Ode94] James J. Odell. "Six Different Kinds of Composition". In: *Journal of Object-Oriented Programming* 5.8 (1994), pp. 10–15.

[OMG11] OMG. *Object Management Group – UML*. Dec. 2011. URL: `http://www.uml.org/`.

[OSG12] OSGi Alliance. *OSGi Core Release 5*. Tech. rep. OSGi Alliance, 2012.

[Pea+04] Russell S. Peak, Joshua Lubell, Vijay Srinivasan, and Stephen C. Waterbury. "STEP, XML, and UML: Complementary Technologies". In: *Journal of Computing and Information Science in Engineering* 4.4 (2004), pp. 379–390.

[Pir74] Robert M. Pirsig. *Zen and the Art of Motorcycle Maintenance*. New York: William Morrow & Company, 1974.

[Ray06] Daniel P. Raymer. *Aircraft Design*. 4th ed. American Institute of Aeronautics and Astronautics, 2006.

[Rec+05] Alan Rector, Chris Welty, Natasha Noy, and Evan Wallace. *Simple part-whole relations in OWL Ontologies*. Tech. rep. W3C, Aug. 11, 2005. URL: `http://www.w3.org/2001/sw/BestPractices/OEP/SimplePartWhole/index.html`.

[Ros08] Stephan Roser. "Designing and Enacting Cross-organisational Business Processes: A Model-driven, Ontology-based Approach". eng. PhD thesis. Universitätsstr. 22, 86159 Augsburg: University of Augsburg, 2008.

[Ros93] Lois Rossetto, ed. *Wired premiere issue*. 1993.

[Sch03] James Schoening. *Standard Upper Ontology Working Group (SUO WG)– Home Page*. Dec. 2003. URL: `suo.ieee.org`.

[Sei12] Arne Seitz. "Advanced Methods for Propulsion System Integration in Aircraft Conceptual Design". PhD thesis. Technische Universität München, 2012.

[Sir+07] Evren Sirin, Bijan Parsia, Bernardo Cuenca Grau, Aditya Kalyanpur, and Yarden Katz. "Pellet: A practical OWL-DL reasoner". In: *Web Semantics: Science, Services and Agents on the World Wide Web* 5.2 (2007), pp. 51–53.

[SMH08] Rob Shearer, Boris Motik, and Ian Horrocks. "HermiT: A Highly-Efficient OWL Reasoner". In: *Proceedings of the 5th International Workshop on OWL: Experiences and Directions (OWLED 2008)*. 2008, pp. 26–27.

[Ste+08] Dave Steinberg, Frank Budinsky, Marcello Paternostro, and Ed Merks. *EMF: Eclipse Modeling Framework*. 2nd ed. Addison Wesley, 2008.

[SZMA11] David Simon Zayas, Anne Monceaux, and Yamine Ait-Ameur. "Using Knowledge and Expressions to Validate Inter-Model Constraints". In: *World Congress*. Vol. 18. 1. 2011, pp. 2737–2742.

[W3C09] W3C. *OWL 2 Web Ontology Language Document Overview*. Oct. 2009. URL: `http://www.w3.org/TR/2009/REC-owl2-overview-20091027/`.

[WCH87] Morton E. Winston, Roger Chaffin, and Douglas Herrmann. “A Taxonomy of Part-Whole Relations”. In: *Cognitive Science* 11.4 (1987), pp. 417 –444.

[Woh+00] Claes Wohlin, Per Runeson, Martin Höst, Magnus C. Ohlsson, Björn Regnell, and Anders Wesselén. *Experimentation in Software Engineering*. Vol. 6. Springer, 2000.

[ZGS11] Sven Ziemer, Martin Glas, and Gernot Stenz. “A Conceptual Design Tool for Multi-Disciplinary Aircraft Design”. In: *Aerospace Conference, 2011 IEEE*. Mar. 2011, pp. 1–13.