DIPLOMAMUNKA

Antal Bálint

Debrecen
2009

Debreceni Egyetem
Informatika kar

ONTOLÓGIÁK A SZOFTVERFEJLESZTÉSBEN

Témavezető:
Dr. Juhász István
egyetemi adjunktus

Készítette:
Antal Bálint
programtervező matematikus

Debrecen
2009

Bevezetés

A tudás kezelésének kérdése régóta meghatározó a szoftverfejlesztési kutatásokban. Ezen kutatások több területet is átölelnek, hiszen a knowledge engineering területének eredményei is jelentősen hozzájárulnak a fejlődéséhez.

Innen származik az ontológia fogalma, amely Gruber(10) szerint egy közös fogalomrendszer formális, explicit specifikációja. Az ontológiák szoftverfejlesztésben történő felhasználásának számos előnye van(4, 5, 8). Mivel egy szakterülethez kapcsolódó koncepcionális modellt ír le, ezért minden eszköztől és környezettől függetlenül újrafelhasználható. Ezt elősegíti az is, hogy nem egy konkrét projekthez íródnak, hanem annál magasabban szinten fogalmazzuk meg azt.

A korábbi megközelítések zárt világot feltételeztek, azaz minden olyan entitás és tulajdonság, amit nem adtunk meg a reprezentációban, nem is létezik. Az ontológiák ezzel szemben nyílt világot reprezentálnak, hiszen az ábrázolt szakterületek is dinamikusak. Végül, az ontológiák általában természetes nyelven (vagy ahhoz közelálló módon) vannak leírva a strukturált információkkal szemben, így megértésük könnyebb, és kevesebb félreértésre adnak okot.

Az ismeretek megosztásán túl lehetőség nyílik a leírt modell validálására is, melynek segítségével annak inkonzisztens voltát bizonyíthatjuk. Dolgozatomban arra teszek kísérletet, hogy az ODMG 3.0-ás szabványhoz(1) készített ontológia segítségével alátámasszam a készülő 4.0-ás szabványt bemutató fehér papír azon állítását, mely szerinte elődje inkonzisztens és nem teljes.

Az ontológia megalkotásához a REFSENO (Representation Formalism for Software Engineering Knowledge, 3) nevű, kifejezetten szoftverfejlesztési ontológiák megalkotására létrehozott formalizmus adaptált változatát használtam. A következőkben a szabvány és a formalizmus rövid bemutatása után ismertetem magát az ontológiát, majd az ennek a segítségével megtalált hibákat.

Az ODMG 3.0 szabvány

A szabvány az ODMG (Object Data Management Group) által kidolgozott specifikációk gyűjteménye, melyek az objektum-orientált programnyelvek objektumainak adatbázisokban történő perzisztens tárolásához adnak keretet. A tárolás úgynevezett ODMS-ekben (Object Data Management System) valósul meg, amelynek két fajtáját különböztetjük meg: az objektumokat közvetlenül tároló ODBMS-ek (Object Database Management System) és az azokat convertáló és relációs vagy egyéb típusú adatbázisban tároló ODM-ek (Obejct-To-Database Mapping).

Az ODMG 3.0 szabvány részei a következők:

- Objektum-modell
- Objektum-specifikációs nyelv
- Objektum-lekérdező nyelv
- Smalltalk, C++, Java nyelvi kötés

Az általam készített ontológia az objektum-modellt dolgozza fel. Az objektum-modell az ODMS-ek számára megadható szemantika fajtáját szabályozza. Többek között az objektumok jellegzetességeit, azok kapcsolatát, elnevezéseit és azonosítását adhatjuk meg.

Napjainkban készül a szabvány 4.0-ás változata(14). Ennek egyik feltevése szerint a 3.0-ás szabvány objektum-modellje inkonzisztens és nem teljes. Dolgozatom célja ennek megmutatása az ontológia segítségével.

A REFSENO

A REFSENO (Representation Formalism for Software Engineering Knowledge) egy formalizmus a szoftverfejlesztési ontológiák létrehozásához. A REFSENO-t 1999-ben hozta létre C. Tautz és C.G. Von Wangenheim olyan céllal, hogy a szoftverfejlesztési tudást konzisztens és könnyen érthető formában tárolhassák.

A félreérthetőség lecsökkentésére alternatív reprezentációkat adhatunk meg: szöveges és grafikus módon is megadhatjuk az ontológiát felépítő elemeket. A formalizmus célja továbbá a szoftverfejlesztési tudás fogalomrendszerének megalkotását explicit módon megadni különböző környezetek és alkalmazás-területek számára. A koncepcionális modellek megosztása is lehetővé válik, ezáltal megkönnyítve azok kommunikálását. A REFSENO segítségével validálhatunk is modelleket.

Egy REFSENO segítségével létrehozott ontológia a következő részekből áll:

- Fogalomszótár: a szoftverfejlesztési entitások és a modellezési okokból bevezetett fogalmak gyűjteménye.
- Terminális attribútumok gyűjteménye: a szoftverfejlesztési entitások tárolását és kinyerését megadó tulajdonságok gyűjteménye.
- Nem-terminális attribútumok gyűjteménye: a szoftverfejlesztési entitások közötti kapcsolatot megadó tulajdonságok gyűjteménye.

A fenti részek megadása táblázatos formában történik, illetve grafikus alternatíva is van ezekre. Az eredeti REFSENO-specifikáció teret biztosít különböző megszorítások megadására a fenti részekkel kapcsolatban. Ezek ismertetésére – terjedelmi okokból – nem térek ki. Ruiz és társai A proposal for a Software Measurement Ontology(13) című cikkükben megadnak egy egyszerűsített formát a REFSENO-hoz. Itt a fogalmak jellemzése nevük, ősük, leírásuk és céljuk, a terminális attribútumoké pedig a fogalom, a név és a leírás,

míg a nem-terminális attribútumoké pedig a név, a kapcsolódó fogalmak és azok leírásának segítségével történik.

Dolgozatomban az utóbbit fogom követni, kiegészítésképpen egy új oszloppal mindegyik esetben, amely a részontológiát fogja megadni. Ezek létrehozására azért volt szükség, mert az ontológia a feldolgozott modell terjedelme miatt nehezen áttekinthető lenne teljes egészében. A bemutatás tehát a részontológiák segítségével történik a továbbiakban, de mellékletként megtekinthető az osztatlan változat is.

Az ontológia

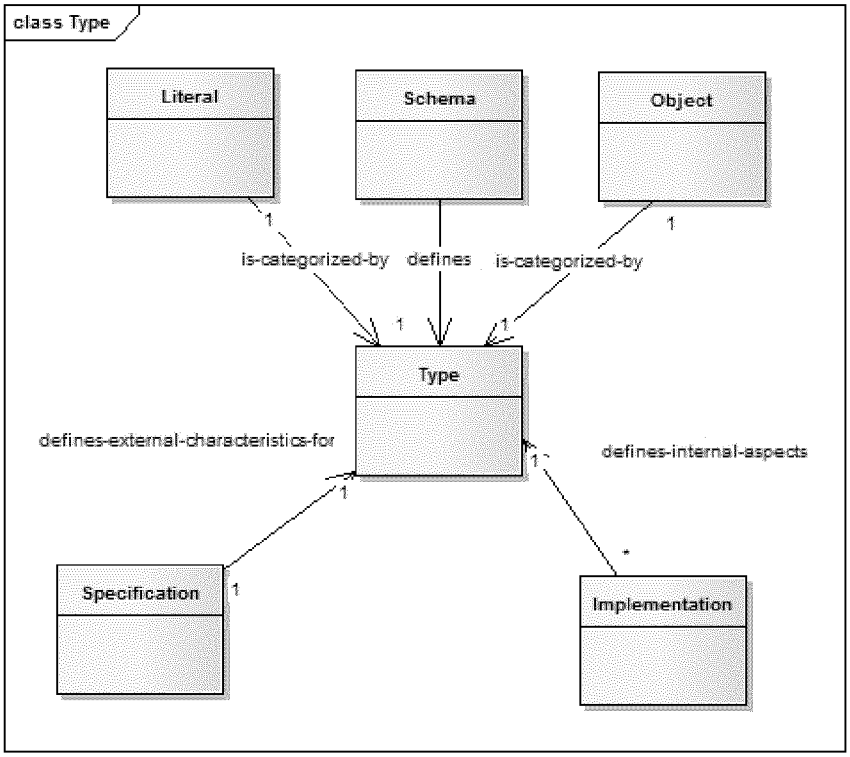
Az ontológiát – terjedelmi okokból - a következő részekre bontottam:

- Típus részontológia
- Objektum részontológia
- Literál részontológia
- ODMS részontológia
- Meta-adat részontológia
- Zár részontológia

Ezek további részeit rendre az adott részontológiánál ismertetem. A bemutatás módja a következő: mindegyik részontológiát egy osztálydiagram segítségével ábrázolom, ezt a REFSENO-féle szöveges leírás követi: a fogalomszótár, a nem-terminális, kapcsolatokat leíró attribútumok, és – amennyiben tartozik az adott részhez – a terminális attribútumok táblázata.

Az ontológia angol nyelven készült, így az eredeti szabvány szövegét fel tudtam használni a definíciók megadásához, ezáltal csökkentve a félreérthetőséget. Minden részhez magyar nyelvű összefoglalót is írtam a könnyebb érthetőség kedvéért.

A típus részontológia



1. ábra: a típus részontológia

A típus részontológia az objektum-modellben megadott típus helyét határozza meg: objektumok és literálok kategorizálására szolgál. Minden típushoz tartozik egy specifikáció és egy vagy több implementáció. A séma adja meg, hogy egy ODMS-en belül milyen típusok adottak. A típus részontológia további részei:

- Specifikációs rész
- Implementációs rész
- Öröklődési rész

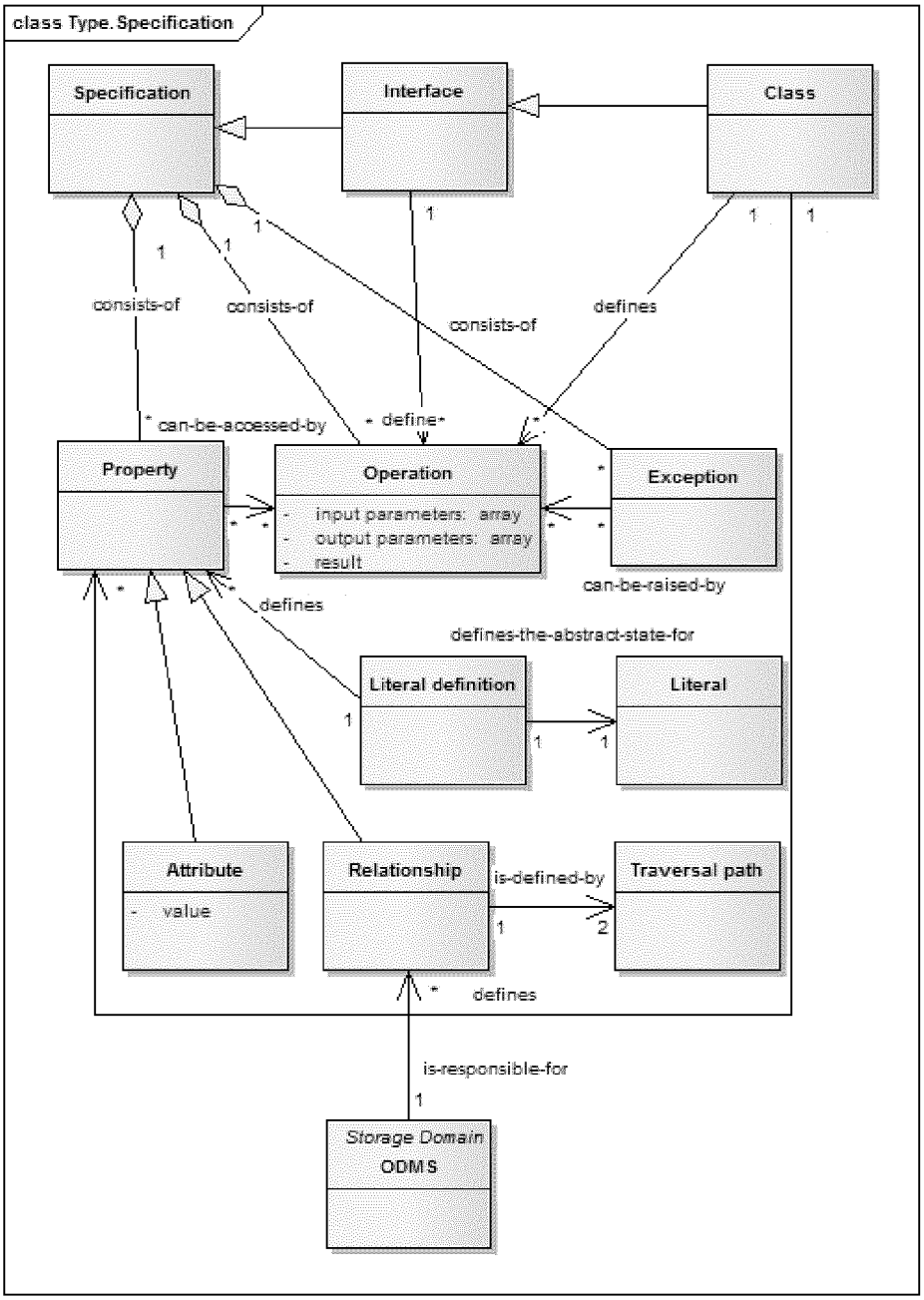
1. Táblázat: A típus részontológia fogalomszótára

Concept	Superconcept	Definition	Subontology
Type	CONCEPT	Types categorizes objects and literals. All elements of a given type have a common range of states (i.e., the same set of properties) and common behavior (i.e., the same set of defined operations). A type has an external specification and one or more implementations.	Type
Specification	CONCEPT	The specification defines the external characteristics of the type. These are the aspects that are visible to users of the type: the operations that can be invoked on its instances, the properties, or state variables, whose values can be accessed, and any exceptions that can be raised by its operations.	Type
Implementation	CONCEPT	A type's implementation defines the internal aspects of the objects of the type: the implementation of the type's operations and other internal details. An implementation of an object type consists of a representation and a set of methods.	Type

2. Táblázat: a típus részontológia nem-terminális attribútumai

Name	Concepts	Description	Subontology
defines-internal-behavior-for	Implementation - Type	A type's implementation defines the internal aspects of the objects of the type.	Type
defines	Schema - Type	An ODMS is based on a schema that is defined in ODL and contains instances of the types defined by its schema.	Type
is-categorized-by	Object - Type	Objects can be categorized by their types.	Type
is-categorized-by	Literal - Type	Literals can be categorized by their types.	Type
defines-external-characteristics-for	Specification - Type	The specification defines the external characteristics of the type	Type

A specifikációs rész



2. ábra: a specifikációs rész

A specifikáció egy típus külső karakterisztikáját adja meg(1. Táblázat): az egyedeken végrehajtható műveletek, az általuk kiváltható kivételek, illetve az objektumok tulajdonságai. Az objektumon kiváltható műveletek a típus viselkedési módját adják meg, míg az állapotát a tulajdonságok által hordozott értékek jelentik. A tulajdonságok két fajtája: az attribútum és a

kapcsolat. A kapcsolatok mindig két típus között definiáltak, és számosságuk szerint lehetnek 1:1, N:1, 1:N és M:N kapcsolatok. A kapcsolatok a bejárási útjuk segítségével definiáltak.

A specifikációnak három típusát különbözteti meg a szabvány: interfész, osztály és literál-definíciós specifikáció. Előbbi egy típus absztrakt viselkedésmódját adja meg. Az osztály egy olyan interfész, amely az állapotát is specifikálja egy típusnak. A literál-definíció egy literáltípus absztrakt állapotát adja meg.

3. Táblázat : a specifikációs rész fogalomszótára

Concept	Superconcept	Definition	Subontology
Property	CONCEPT	The values carried by a set of properties defines the state of an object.	Type.Specification
Attribute	Property	Defines the abstract state of an object.	Type.Specification
Relationship	Property	A relationship is defined between two types, each of which must have instances that are referenceable by object identifiers. The ODMS is responsible for maintaining the referential integrity of relationships. The implementation of relationships is encapsulated by public operations that form and drop members from the relationship, plus public operations on the relationship target classes to provide access and to manage the required referential integrity constraints.	Type.Specification
Operation	CONCEPT	The behavior of an object is defined by the set of operations that can be executed on or by the object. An operation has a name, which can be overloaded. Operations may have a list of input and output parameters, each with a specified type. Each operation may also return a typed result. Some operations may return no value. An operation may have side effects. The Object Model assumes sequential execution of operations.	Type.Specification

Exception	CONCEPT	Operations can raise exceptions, and exceptions can communicate exception results. Mappings for exceptions are defined by each language binding.	Type.Specification
Interface	Specification	An interface definition is a specification that defines only the abstract behavior of an object type.	Type.Specification
Class	Specification	A class definition is a specification that defines the abstract behavior and abstract state of an object type. A class is an extended interface with information for ODMS schema definition.	Type.Specification
Literal definition	Specification	A literal definition defines only the abstract state of a literal type.	Type.Specification
Traversal path	CONCEPT	A relationship is defined explicitly by declaration of traversal paths that enable applications to use the logical connections between the objects participating in the relationship. Traversal paths are declared in pairs, one for each direction of traversal of the relationship.	Type.Specification

4. Táblázat: a specifikációs rész terminális attribútumai

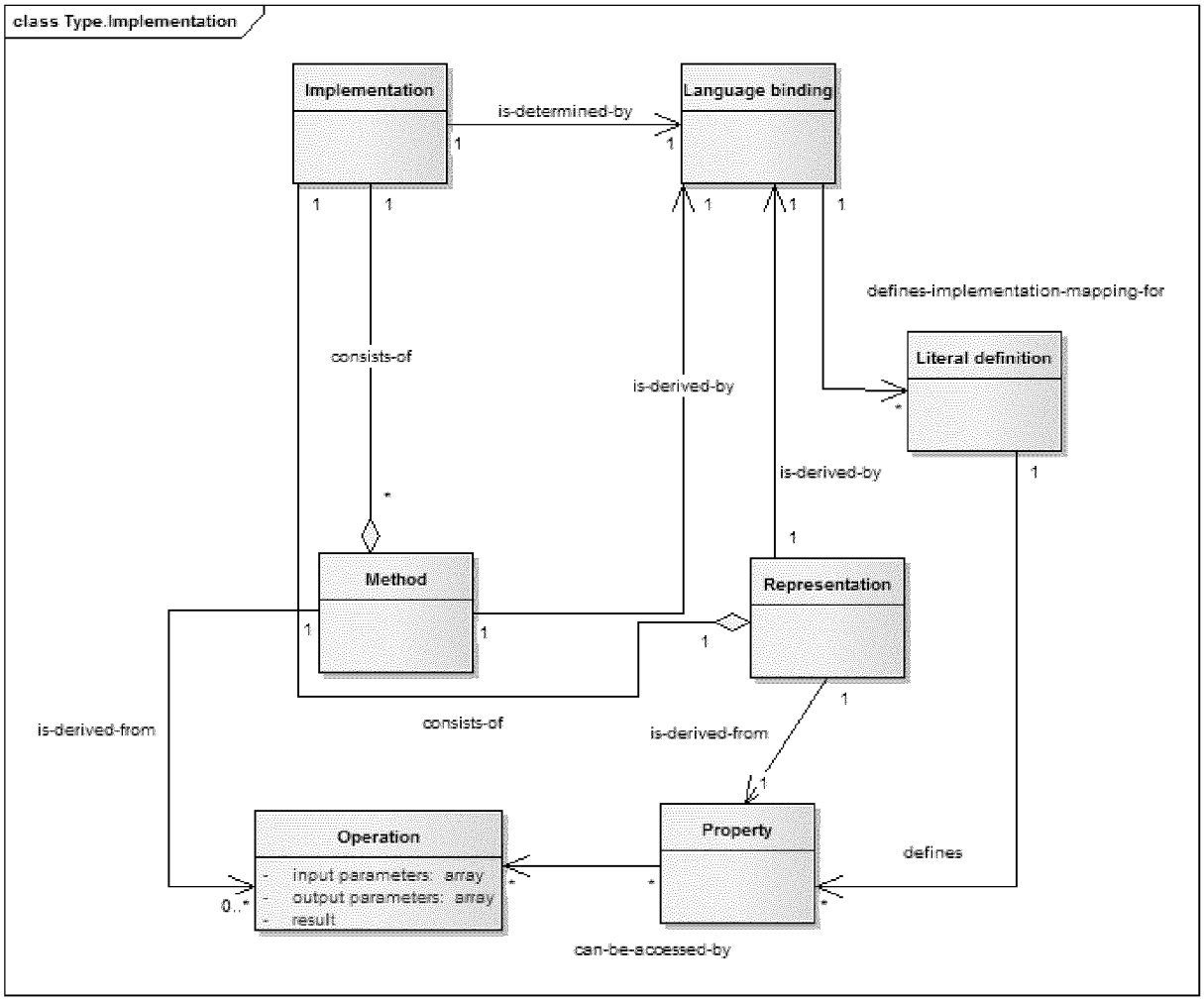
Concept	Attribute	Description	Cardinality	Subontology
Attribute	value	Attribute has a value.	1	Type.Specification
Operation	input parameter	Operations may have a list of input and output parameters, each with a specified type.	*	Type.Specification
	output paramater	Operations may have a list of input and output parameters, each with a specified type.	*	
	result	Each operation may also return a typed result. Some operations may return no value.	1	

5. Táblázat: a specfikiációs rész nem-terminális attribútumai

Name	Concepts	Description	Subontology
is-a	Interface - Specification	Interface is a specification.	Type.Specification
is-a	Class - Interface	Class is an interface.	Type.Specification
consists-of	Specification - Property	An external specification of a type consists of an implementation-independent, abstract description of the operations, exceptions, and properties that are visible to users of the type.	Type.Specification
consists-of	Specification - Operation	An external specification of a type consists of an implementation-independent, abstract description of the operations, exceptions, and properties that are visible to users of the type.	Type.Specification
consists-of	Specification - Exception	An external specification of a type consists of an implementation-independent, abstract description of the operations, exceptions, and properties that are visible to users of the type.	Type.Specification
defines	Interface - Property	Interface defines abstract state.	Type.Specification
defines	Class - Operation	Class defines abstract behavior.	Type.Specification
defines	Literal definition - Property	Literal definition defines abstract state.	Type.Specification
can-be-accessed-by	Property - Operation	Properties can be accessed by operations.	Type.Specification
can-be-raised-by	Exception - Operation	Exceptions can be raised by operations.	Type.Specification
is-a	Property - Attribute	Attribute is a Property.	Type.Specification
is-a	Relationship - Attribute	Relationship is a Property.	Type.Specification
is-defined-by	Relationship - Traversal Path	A relationship is defined explicitly by declaration of traversal paths that enable applications to use the logical connections between the objects participating in the relationship. Traversal paths are declared in pairs, one for each direction of traversal of the relationship.	Type.Specification
is-responsible-for	ODMS - Relationship	The ODMS is responsible for maintaining the referential integrity of relationships.	Type.Specification

defines-the-abstract-state-for	Literal definition - Literal	A literal definition defines only the abstract state of a literal type.	Type.Specification
is-a	Literal definiton - Specification	Literal definiton is a specification.	Type.Specification

Az implementációs rész



Az implementációs rész a típus belső megvalósítását szabályozza. Az implementáció egy reprezentációból és metódusok egy halmazából épül fel. Az implementációt a nyelvi kötés határozza meg. A reprezentáció a típus absztrakt állapotából származó adastruktúra, amely a nyelvi kötés által jön létre. A metódusokat a típus absztrakt viselkedés módjából

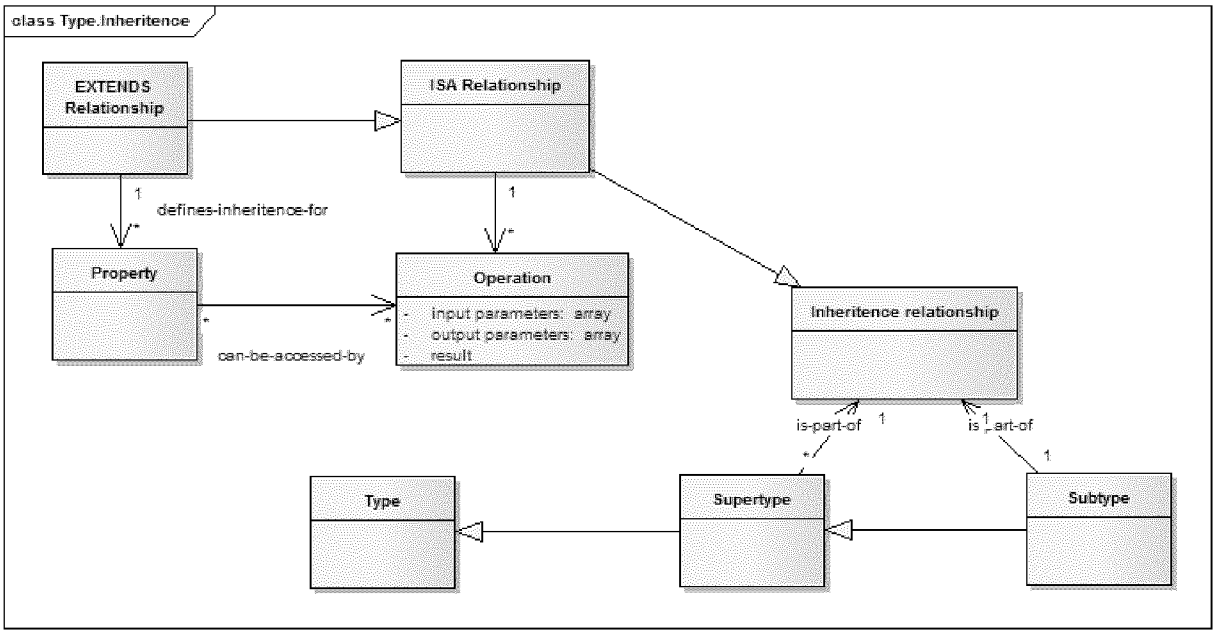
származtatja a nyelvi kötés. A nyelvi kötés a literál-definíciók számára implementációs leképezést valósít meg.

Concept	Superconcept	Definition	Subontology
Language binding	CONCEPT	The implementation of a type is determined by a language binding.	Implementation
Representation	CONCEPT	The representation is a data structure that is derived from the type's abstract state by a language binding: For each property contained in the abstract state there is an instance variable of an appropriate type defined.	Implementation
Method	CONCEPT	The methods are procedure bodies that are derived from the type's abstract behavior by the language binding: For each of the operations defined in the type's abstract behavior a method is defined. This method implements the externally visible behavior of an object type. A method might read or modify the representation of an object's state or invoke operations defined on other objects. There can also be methods in an implementation that have no direct counterpart to the operations in the type's specification.	Implementation

Name	Concepts	Description	Subontology
is-determined-by	Implementation - Language binding	The implementation of a type is determined by a language binding.	Implementation
defines-implementation-mapping-for	Language binding - Literal defintion	Each language binding also defines an implementation mapping for literal types.	Implementation
consists-of	Implementation - Representation	An implementation of an object type consists of a representation and a set of methods.	Implementation
consists-of	Implementation - Method	An implementation of an object type consists of a representation and a set of methods.	Implementation
derived-by	Representation - Language binding	Representation is derived by the language binding from	Implementation

		the type's abstract state.	
derived-by	Method - Language binding	Methods are derived by the language binding from the type's abstract behavior.	Implementation
is-derived-from	Representation - Property	Representation is derived by the language binding from the type's abstract state.	Implementation
is-derived-from	Method - Operation	Methods are derived by the language binding from the type's abstract behavior.	Implementation

Az öröklődési rész



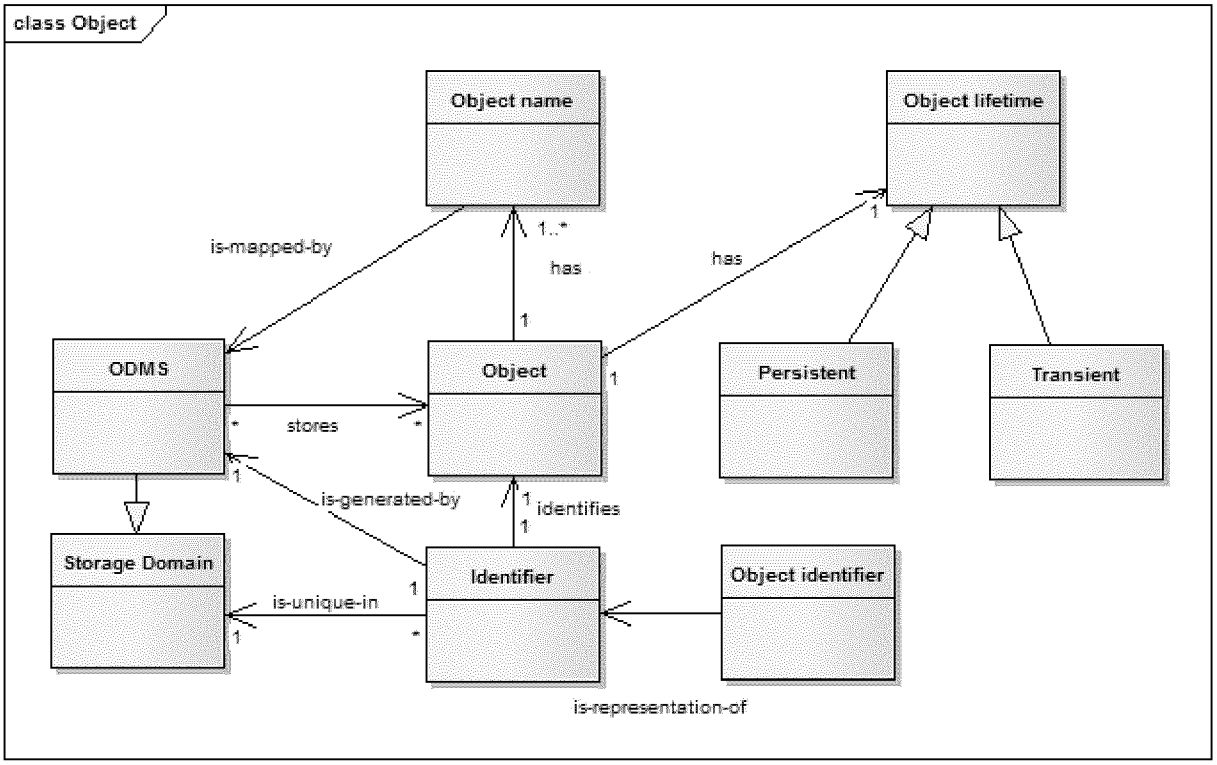
Az öröklődési rész az objektum-modell öröklődési szabályait mutatja be. Egy öröklődési kapcsolatban mindig két típus vesz részt: az általánosabb, a hierarchiában fentebb elhelyezkedőt őstípusnak, a specifikusabb, a hierarchiában alacsonyabb szinten lévő típust altípusnak nevezzük. Az öröklődési kapcsolat két típusa az ISA-kapcsolat és az EXTENDS. Előbbi csak az absztrakt viselkedési mód számára definiál öröklődést, míg utóbbi olyan speciális ISA-kapcsolatnak tekinthető, amely az absztrakt állapot számára is biztosítja ezt.

Concept	Superconcept	Definition	Subontology
Inheritence relationship	CONCEPT	A relationship between two types, when one type (the subtype) inherits some characteristics of the other (the supertype).	Inheritence
ISA relationship	Inheritence relationship	Defines the inheritence of behavior between two types.	Inheritence
EXTENDS relationship	Inheritence relationship	Defines the inheritence of behavior and state between two types. Applies only to object types.	Inheritence
Supertype	Type	Is a part of the inheritence relationship and is a type.	Inheritence

Subtype	Supertype	Is a part of the inheritence relationship and is a type same as it supertype's with additional abstract state or behavior.	Inheritence

Name	Concepts	Description	Subontology
is-a	Supertype - Type	Supertype is a type.	Inheritence
is-a	Subtype - Supertype	Subtype is a supertype.	Inheritence
is-part-of	Supertype - Inheritence relationship	Supertype is a part of an inheritence relationship.	Inheritence
is-part-of	Subtype - Inheritence relationship	Subtype is a part of an inheritence relationship.	Inheritence
is-a	ISA relationship - Inheritence relationship	ISA relationship is an inheritence relationship.	Inheritence
is-a	EXTENDS relationship - ISA relationship	EXTENDS relationship is an ISA relationship.	Inheritence
defines-inheritence-for	ISA relationship - Operation	ISA defines the inheritance of behavior.	Inheritence
defines-inheritence-for	EXTENDS relationship - Property	EXTENDS defines the inheritance of behavior and state.	Inheritence

Az objektum részontológia



Az objektum részontológia az objektumok jellemzésére szolgál. Az objektumok egy vagy több névvel rendelkeznek, melyek objektumhoz történő társítását az ODMS végzi. Az objektumoknak van élettartama, amely lehet tranziens vagy perzisztens. Előbbi esetben a foglalt memória területet a program futtató környezete, utóbbit az ODMS futásidejű alrendszere kezeli. Az objektumok azonosítása azonosító segítségével történik, melynek reprezentációja az objektumazonosító. Generálását az ODMS végzi, egyedisége pedig a tárolási területre korlátozódik.

Az objektum részontológia további részei:

- Objektumtípusok rész
- Kulcs rész

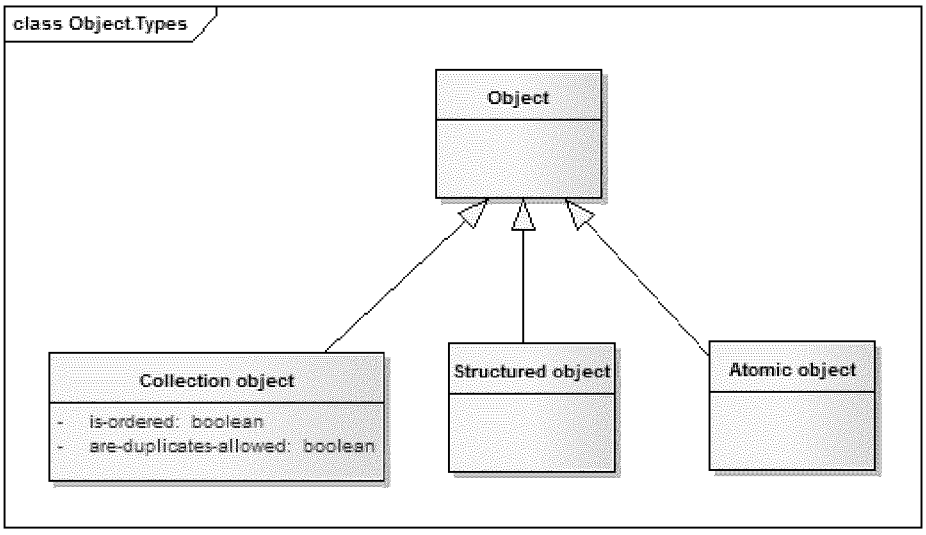
Concept	Superconcept	Definition	Subontology

Object	CONCEPT	Object is a basic modelling primitive. Each object has a unique identifier, a state and a behavior. An object is sometimes referred to as an instance of its type.	Object
Identifier	CONCEPT	Identifies an object in a storage domain.	Object
Object identifier	Representation	The representation of the identity of an object is referred to as its object identifier. An object retains the same object identifier for its entire lifetime. Object identifiers are generated by the ODMS.	Object
Object name	CONCEPT	In addition to being assigned an object identifier by the ODMS, an object may be given one or more names that are meaningful to the programmer or end user.	Object
Object lifetime	CONCEPT	The lifetime of an object determines how the memory and storage allocated to the object are managed. The lifetime of an object is specified at the time the object is created.	Object
Persistent	Object lifetime	An object whose lifetime is persistent is allocated memory and storage managed by the ODMS runtime system.	Object
Transient	Object lifetime	An object whose lifetime is transient is allocated memory that is managed by the programming language runtime system.	Object

Name	Concepts	Description	Subontology
has	Object - Object name	Object has one or more object name	Object
has	Object - Object lifetime	Object has a lifetime.	Object
stores	ODMS - Object	ODMS stores objects	Object
identifies	Identifier - Object	All identifiers of objects in an ODMS are unique, relative to each other.	Object
is-generated-by	Identifier - ODMS	Object identifiers are generated by the ODMS.	Object
is-unique-in	Identifier - Storage domain	Because all objects have identifiers, an object can always be distinguished from all other	Object

		objects within its storage domain.	
is-representation-of	Object identifier - Identifier	The representation of the identity of an object is referred to as its object identifier.	Object
is-a	Persistent - Object lifetime	Persistent is a lifetime.	Object
is-a	Transient - Object lifetime	Transient is a lifetime.	Object
mapped-by	Object name - ODMS	The ODMS provides a function that it uses to map from an object name to an object.	Object

Az objektumtípusok rész



Az objektumtípusok három típusfajtát mutatna be: a kollekciókat, a strukturált objektumokat, és az atomi objektumokat. További részei:

- Kollekciók rész
- Strukturált objektum rész

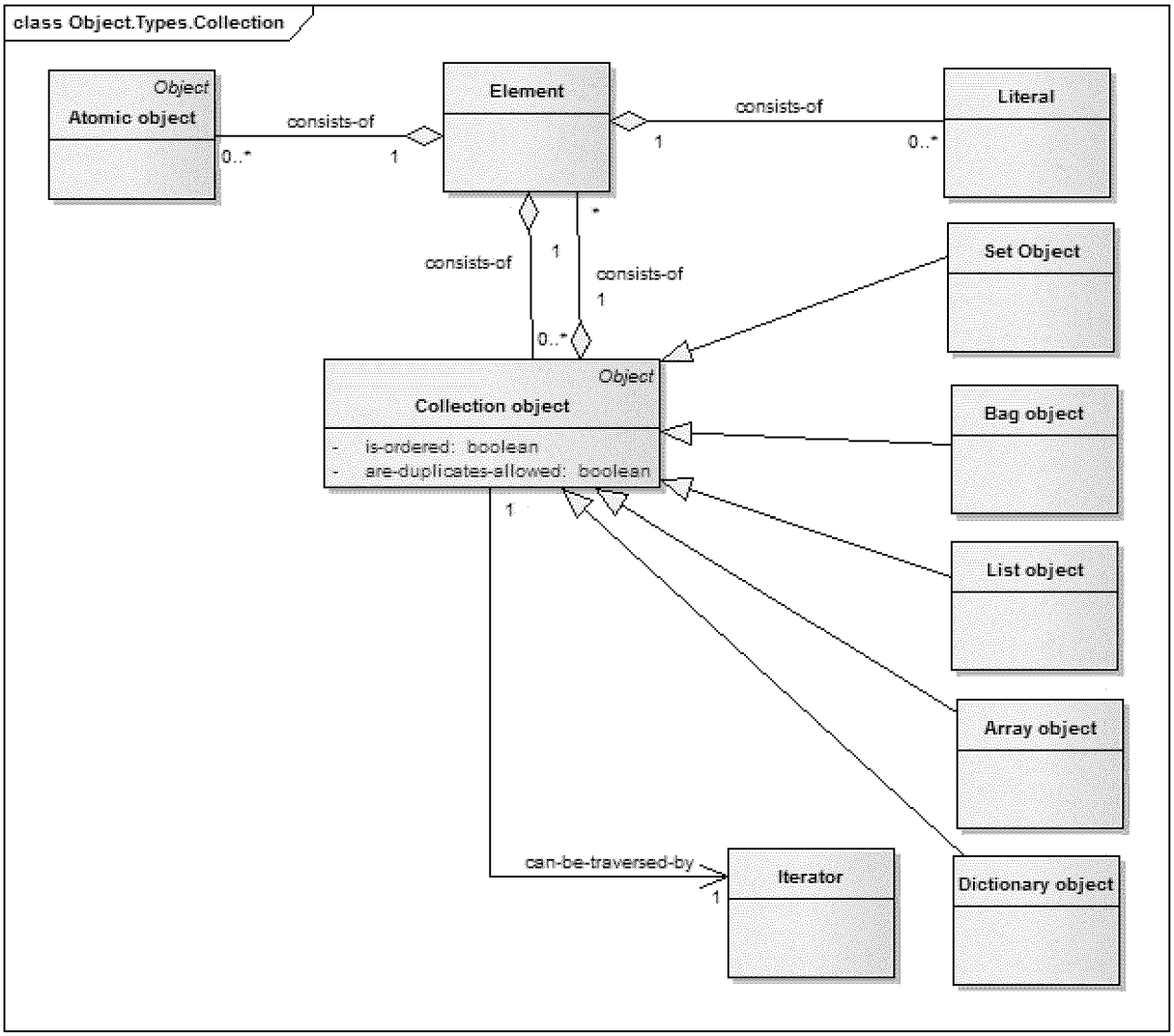
Concept	Superconcept	Definition	Subontology
Atomic object	Object	Atomic object is an object which is user-defined.	Object.Types

Collection object	Object	In the ODMG Object Model, instances of collection objects are composed of distinct elements, each of which can be an instance of an atomic type, another collection, or a literal type. All the elements of the collection must be of the same type.	Object.Types
Structured object	Object	All structured objects support the Object ODL interface.	Object.Types

Concept	Attribute	Description	Cardinality	Subontology
Collection object	is-ordered	Declares the orderedness of the object.	1	Object.Types
	are-duplicates-allowed	Declares whether the duplicates allowed.	1	

Name	Concepts	Description	Subontology
is-a	Atomic object - Object	An atomic object type is an user-defined object.	Object.Types
is-a	Collection object - Object	Collection object is an object.	Object.Types
is-a	Structured object - Object	Structured object is an object.	Object.Types

A kollekciók rész



A kollekciók különálló elemekből álló objektumok. Ezek az elemek lehetnek atomi objektumok, literálok, vagy egyéb kollekciók. Egy kollekció elemei azonos típussal kell rendelkezzenek. A kollekciók bejárása iterátor segítségével történik. A kollekciók fajtái: lista, halmaz, tömb, szótár és táska. A kollekciókat jellemzi rendezettségük, és az, hogy duplikált elemek engedélyezettek-e számára.

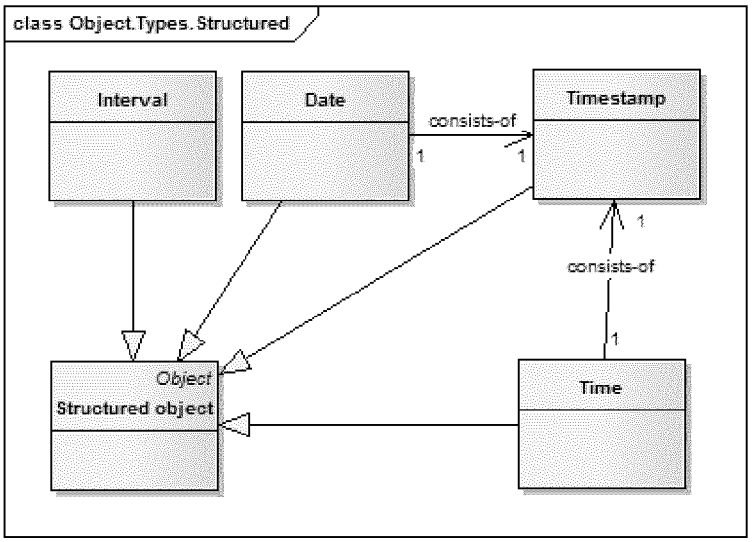
Concept	**Superconcept**	**Definition**	**Subontology**
Iterator	CONCEPT	An iterator is a mechanism for accessing the elements	Object.Types.Collection

		of a Collection object, can be created to traverse a collection.	
Set object	Collection objects	A Set object is an unordered collection of elements, with no duplicates allowed.	Object.Types.Collection
Bag object	Collection objects	A Bag object is an unordered collection of elements that may contain duplicates.	Object.Types.Collection
List object	Collection objects	A List object is an ordered collection of elements.	Object.Types.Collection
Array object	Collection objects	An Array object is a dynamically sized, ordered collection of elements that can be located by position.	Object.Types.Collection
Dictionary object	Collection objects	A Dictionary object is an unordered sequence of key-value pairs with no duplicate keys.	Object.Types.Collection
Element	CONCEPT	For modelling reasons: element of a collection. Can be an instance of an atomic type, another collection, or a literal type. All the elements of the collection must be of the same type.	Object.Types.Collection

Name	Concepts	Description	Subontology
consists-of	Collection object - Element	In the ODMG Object Model, instances of collection objects are composed of distinct elements, each of which can be an instance of an atomic type, another collection, or a literal type.	Object.Types.Collection
consists-of	Element - Atomic object	In the ODMG Object Model, instances of collection objects are composed of distinct elements, each of which can be an instance of an atomic type, another collection, or a literal type.	Object.Types.Collection

consists-of	Element - Collection object	In the ODMG Object Model, instances of collection objects are composed of distinct elements, each of which can be an instance of an atomic type, another collection, or a literal type.	Object.Types.Collection
consists-of	Element - Literal	In the ODMG Object Model, instances of collection objects are composed of distinct elements, each of which can be an instance of an atomic type, another collection, or a literal type.	Object.Types.Collection
is-a	Set object - Collection object	Set object is a collection object.	Object.Types.Collection
is-a	Bag object - Collection object	Bag object is a collection object.	Object.Types.Collection
is-a	List object - Collection object	List object is a collection object.	Object.Types.Collection
is-a	Array object - Collection object	Array object is a collection object.	Object.Types.Collection
is-a	Dictionary object - Collection object	Dictionary object is a collection object.	Object.Types.Collection
can-be-traversed-by	Collection object - Iterator	An iterator is a mechanism for accessing the elements of a Collection object, can be created to traverse a collection.	Object.Types.Collection

A strukturált objektum rész

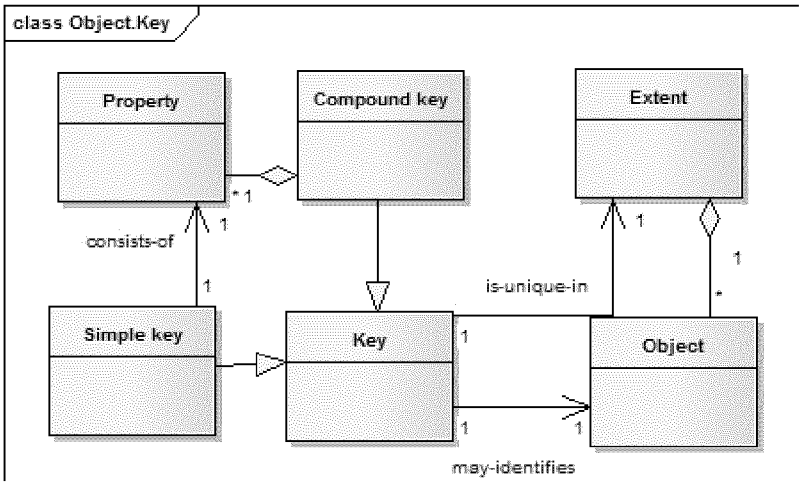


A strukturált objektumok fajtái: dátum, intervallum, idő és időbélyeg (mely egy dátum- és egy idő-objektumból áll).

Concept	Superconcept	Definition	Subontology
Date	Structured objects	A structured object defined by the DateFactory interface and the Date class.	Object.Types.Structured
Interval	Structured objects	Intervals represent a duration of time and are used to perform some operations on Time and Timestamp objects.	Object.Types.Structured
Time	Structured objects	Times denote specific world times, which are internally stored in Greenwich Mean Time (GMT).	Object.Types.Structured
Timestamp	Structured objects	Timestamps consist of an encapsulated Date and Time.	Object.Types.Structured

Name	Concepts	Description	Subontology
is-a	Date - Structured object	Date is a structured object.	Object.Types.Structured
is-a	Interval - Structured object	Interval is a structured object.	Object.Types.Structured
is-a	Time - Structured object	Time is a structured object.	Object.Types.Structured
is-a	Timestamp - Structured object	Timestamp is a structured object.	Object.Types.Structured
consists-of	Timestamp - Date	Timestamps consist of an encapsulated Date and Time.	Object.Types.Structured
consists-of	Timestamp - Time	Timestamps consist of an encapsulated Date and Time.	Object.Types.Structured

A kulcs rész



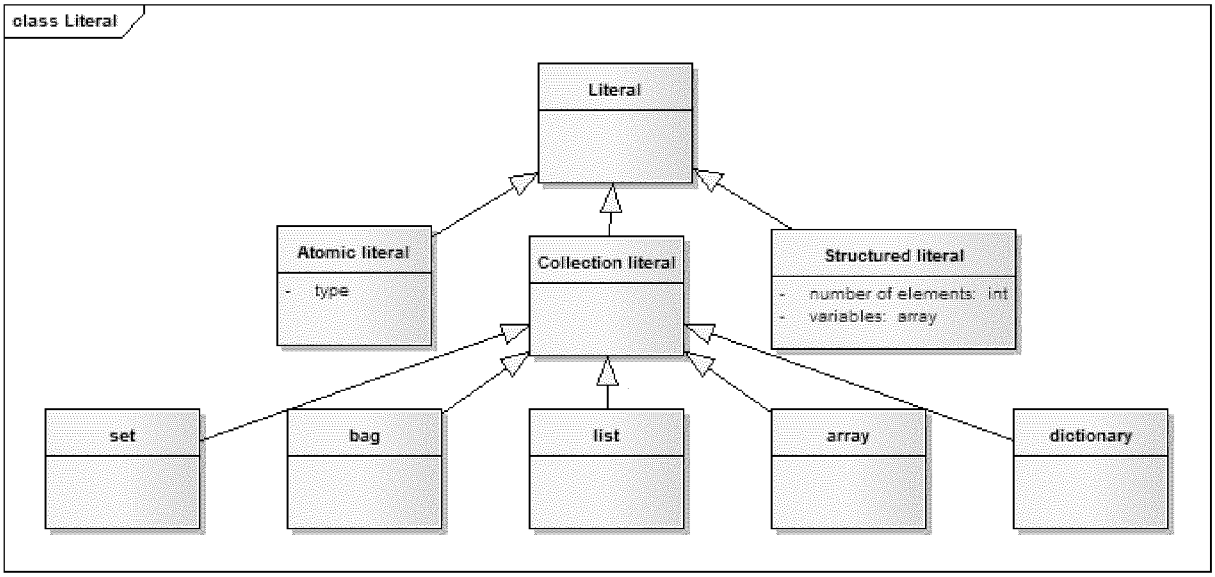
A kulcsok a tulajdonságok olyan csoportja, amely alkalmas egy objektum azonosítására. Egyszerű kulcsról beszélünk, ha egy tulajdonságból áll; több tulajdonság esetén összetett kulcs a megnevezése. A kulcs egyediségének területe a kiterjedése.

Concept	Superconcept	Definition	Subontology
Key	CONCEPT	Keys are some property or set of properties which can identifiy the individual objects by the values they carry in some cases.	Object.Key
Simple key	Key	A simple key is a key consists of a single property.	Object.Key
Compund key	Key	A compound key is a key consists of a set of properties.	Object.Key

Name	Concepts	Description	Subontology
may-identifies	Key - Object	In some cases, the individual instances of a type can be uniquely identified by the values they carry for some property or set of properties. These identifying properties are called keys.	Object.Key

is-unique-in	Key - Extent	The scope of uniqueness is the extent of the type; thus, a type must have an extent to have a key.	Object.Key
is-a	Simple key - Key	Simple key is a key.	Object.Key
consists-of	Simple key - Property	A simple key consists of a single property.	Object.Key
is-a	Compound key - Key	Compound key is a key.	Object.Key
consists-of	Compound key - Property	A compund key consists of a set of properties.	Object.Key

A literál részontológia



A literál az objektum mellett a másik modellezési primitív. Típusai: az atom literál, a strukturált literál és a kollekcióliterál. Utóbbi analóg az objektumoknál tárgyaltakkal. A strukturált literál fix elemszámú, melyek mindegyike rendelkezik egy változónévvel.

Concept	Superconcept	Definition	Subontology
Literal	CONCEPT	Literal is a basic modelling primitive. Literals can be categorized by their types.A literal has no identifier.	Literal
Atomic literal	Literal Types	Numbers and characters are examples of atomic literal types. Instances of these types are not explicitly created by applications, but rather implicitly exist.	Literal
Collection literal	Literal Types	Analogues to the collection objects.	Literal
Structured literal	Literal Types	A structured literal, or simply structure, has a fixed number of elements, each of which has a variable name and can contain	Literal

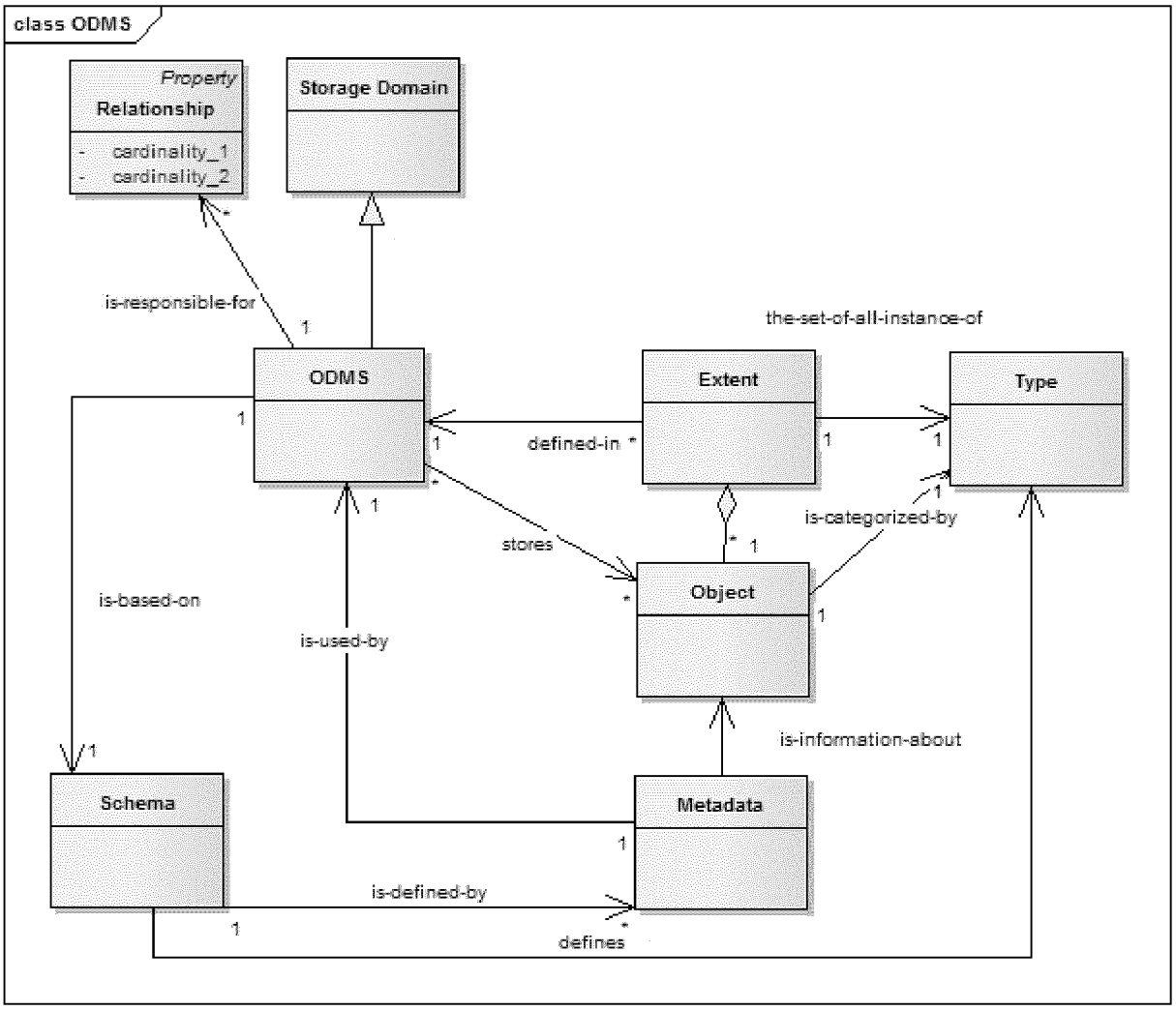
		either a literal value or an object.	
set	Collection literal	Analogous to Set object.	Literal
bag	Collection literal	Analogous to Bag object.	Literal
list	Collection literal	Analogous to List object.	Literal
array	Collection literal	Analogous to Array object.	Literal
dictionary	Collection literal	Analogous to Dictionary object.	Literal

Concept	Attribute	Description	Cardinality	Subontology
Atomic literal	type	The ODMG Object Model supports the following types of atomic literals: • long • long long • short • unsigned long • unsigned short • float • double • boolean • octet • char (character) • string • enum (enumeration)	1	Literal
Structured literal	number of elements	A structured literal, or simply structure, has a fixed number of elements, each of which has a variable name and can contain either a literal value or an object.	1	Literal
	variables	A structured literal, or simply structure, has a fixed number of elements, each of which has a variable name and can contain either a literal value or an object.	*	

Name	Concepts	Description	Subontology
is-a	Atomic literal - Literal	Atomic literal is a literal.	Literal
is-a	Collection literal - Literal	Collection literal is a literal.	Literal
is-a	set - Collection literal	set is a collection literal.	Literal
is-a	bag - Collection literal	bag is a collection literal.	Literal
is-a	list - Collection literal	list is a collection literal.	Literal

is-a	array - Collection literal	array is a collection literal.	Literal
is-a	dictionary - Collection literal	dictionary is a collection literal.	Literal
is-a	Structured literal - Literal	Structured literal is a literal.	Literal

Az ODMS részontológia



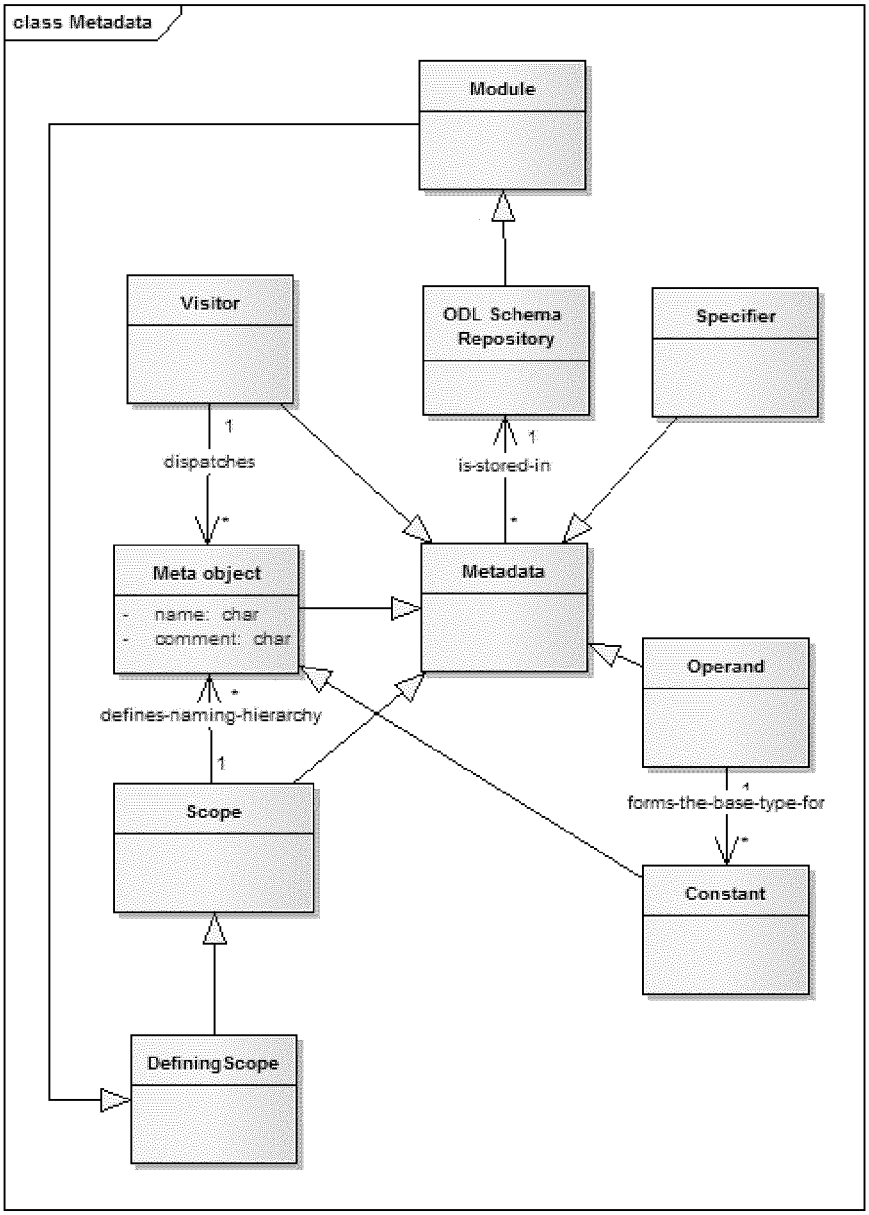
Az ODMS egy tárolási terület, mely objektumok tárolására szolgál és a sémán alapul. A séma az objektum tárolásának struktúrájának megadásához meta-adatokat használ. A meta-adat objektumokról szóló információ. A séma által megadott típusok összes példányát kiterjedésnek nevezzük, mely egy ODMS-en belül definiált.

Concept	Superconcept	Definition	Subontology
ODMS	CONCEPT	ODMS is Object Data Management System. An ODMS stores objects, enabling them to be shared by multiple users and applications.	ODMS
Schema	CONCEPT	An ODMS is based on a schema that is defined in ODL and contains instances of the types defined by its schema.	ODMS
Extent	CONCEPT	The extent of a type is the set of all instances of the type within a particular ODMS.	ODMS
Storage domain	CONCEPT		ODMS
Metadata	CONCEPT	Metadata is descriptive information about persistent objects that defines the schema of an ODMS. Metadata is used by the ODMS to define the structure of its object storage, and at runtime, guides access to the ODMS's persistent objects.	ODMS

Name	Concepts	Description	Subontology
defines	Schema - Type	An ODMS is based on a schema that is defined in ODL and contains instances of the types defined by its schema.	ODMS
is-based-on	ODMS - Schema	ODMS is based on a schema.	ODMS
the-set-of-all-instances-in	Extent - Type	The extent of a type is the set of all instances of the type within a particular ODMS.	ODMS
defined-in	Extent - ODMS	The extent of a type is the set of all instances of the type within a particular ODMS.	ODMS
consists-of	Extent - Object	The extent of a type is the set of all instances of the type within a particular ODMS.	ODMS
is-information-about	Metadata - Object	Metadata is descriptive information about persistent objects that defines the schema of an ODMS.	ODMS
is-defined-by	Schema - Metadata	Metadata is descriptive information about persistent objects that defines the schema of an ODMS.	ODMS

is-used-by	Metadata - ODMS	Metadata is used by the ODMS to define the structure of its object storage, and at runtime, guides access to the ODMS's persistent objects.	ODMS
is-a	ODMS - Storage domain	ODMS is a storage domain.	ODMS

A meta-adat részontológia



A meta-adatok az ODL sématárolóban helyezkednek el A meta-adatok közé tartoznak a metaobjektumok, az érvényességi tartomány, a látogató, az operandus és a specifikáló. Az érvényességi tartományok névadási hierarchiát adnak meg a metaobjektumok számára. A definiáló érvényességi tartományok ezek speciális esetei, amelyek más metaobjektumokat is tartalmazhatnak, azokat kezelni tudják operátoraik segítségével. A modulok ezek speciális esetei, amelyben a definiálás a modulokra vonatkozik. Ezek közé tartozik az ODL sématároló is. A látogatók a meta-objektumok tárolóbeli bejárásához biztosítanak kapcsolási

mechanizmus. A konstansok segítségével statikus módon rendelhetünk nevekhez értékeket a tárolóban. Az operandusok ehhez biztosítanak alaptípust.

Concept	Superconcept	Definition	Subontology
ODL Schema Repository	Modul	Metadata is stored in an ODL Schema Repository, which is also accessible to tools and applications using the same operations that apply to user-defined types.	Metadata
Meta object	Object	Subclasses of MetaObject interface.	Metadata
Scope	CONCEPT	Scopes define a naming hierarchy for the meta objects in the repository.	Metadata
DefiningScope	Scope	DefiningScopes are Scopes that contain other meta object definitions using their defines relationship and that have operations for creating, adding, and removing meta objects within themselves.	Metadata
Visitor	CONCEPT	Visitors provide a convenient "double dispatch" mechanism for traversing the meta objects in the repository.	Metadata
Modul	Meta object, DefiningScope	Modules are DefiningScopes that define operations for creating modules and interfaces within themselves.	Metadata
Specifier	Interface	Specifiers are used to assign a name to a type in certain contexts.	Metadata
Operand	Interface	Operands form the base type for all constant values in the repository.	Metadata
Constants	Meta object	Constants provide a mechanism for statically associating values with names in the repository. The value is defined by an Operand subclass that is either a literal value (Literal), a reference to another Constant (ConstOperand), or the value of a constant expression (Expression). Each constant has an associated type and keeps track of the other ConstOperands that refer	Metadata

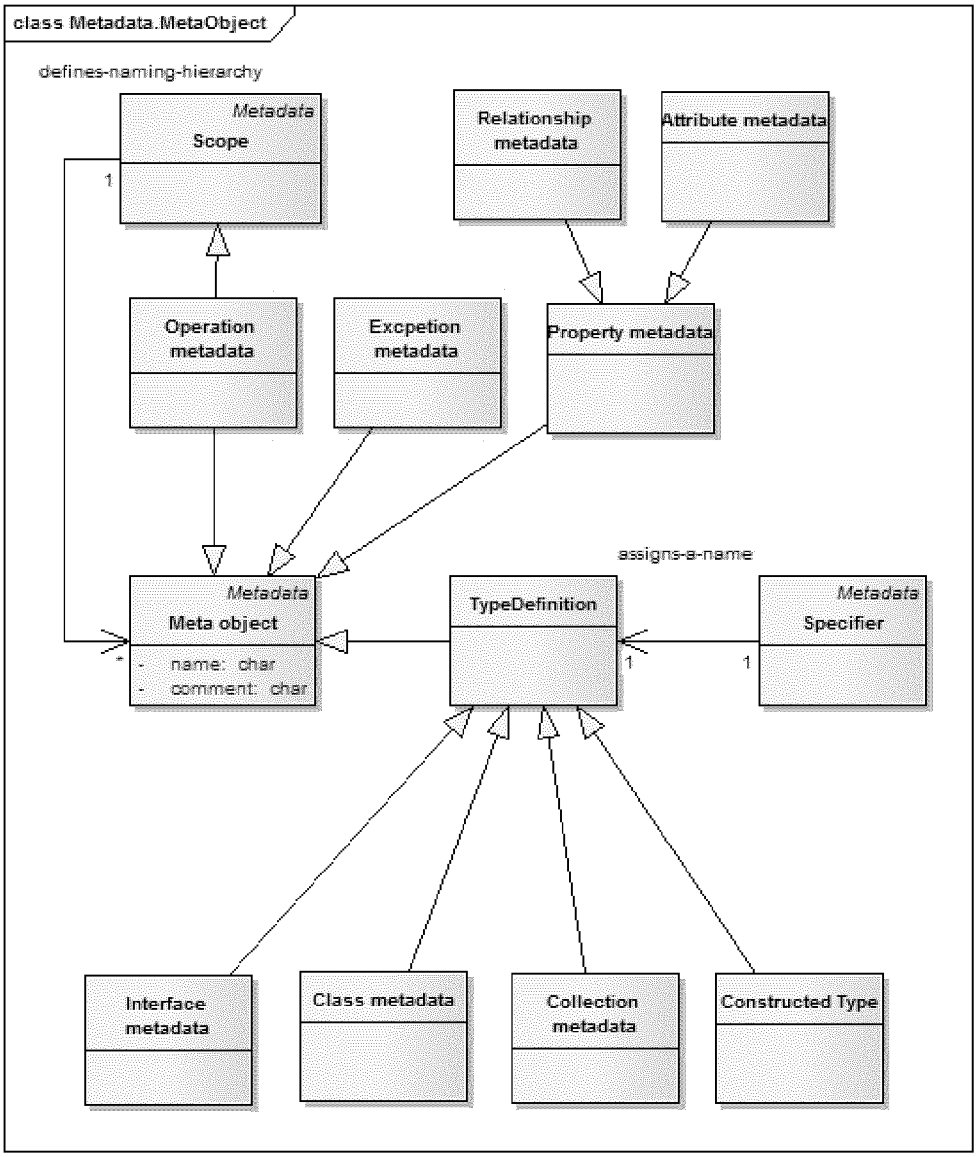
		to it in the repository. The value operation allows the constant's actual value to be computed at any time.	

Concept	Attribute	Description	Cardinality	Subontology
Meta object	name	Meta objects have names.	1	Metadata
	comment	Meta objects stores comments.	1	

Name	Concepts	Description	Subontology
is-stored-in	Metadata - ODL Schema Repository	Metadata is stored in an ODL Schema Repository.	Metadata
is-a	Meta object - Metadata	Meta object is a metadata.	Metadata
is-a	Scope - Metadata	Scope is a metadata.	Metadata
is-a	Visitor - Metadata	Visitor is a metadata.	Metadata
is-a	Operand - Metadata	Operand is a metadata.	Metadata
is-a	Specifier - Metadata	Specifier is a metadata.	Metadata
defines-naming-hierarchy-for	Scope - Meta object	Scopes define a naming hierarchy for the meta objects in the repository.	Metadata
is-a	DefiningScope - Scope	DefiningScopes are Scopes that contain other meta object definitions using their defines relationship and that have operations for creating, adding, and removing meta objects within themselves.	Metadata
dispatches	Visitor - Meta object	Visitors provide a convenient "double dispatch" mechanism for traversing the meta objects in the repository.	Metadata
is-a	Module - DefiningScope	Modules and the Schema Repository itself, which is a specialized module, are Defining-Scopes that define operations for creating modules and interfaces within themselves.	Metadata
is-a	ODL Schema Repository - Module	Modules and the Schema Repository itself, which is a specialized module, are Defining-Scopes that define operations for	Metadata

		creating modules and interfaces within themselves.	
is-a	Constant - Meta object	Constant is a meta object.	Metadata
forms-the-base-type-for	Operand - Constant	Operands forms the base type for constants.	Metadata

A metaobjektum rész



A metaobejktumok közé tartoznak a műveletekhez, kivételekhez és tulajdonságokhoz tartozó meta-adatok. Utóbbinak két fajtája van: a kapcsolati és az attribútum-meta-adatok. A típsudefiníciók új nevekez rendelnek az átalaluk hivatkozott típusokhoz. A specfikiálók nevedet rendelnek a típusokhoz egy bizonyos környezetben. A típusdefiníciók fajtái: interfész, osztály, kollekció és a származtatott típusokról szóló meta-adatok.

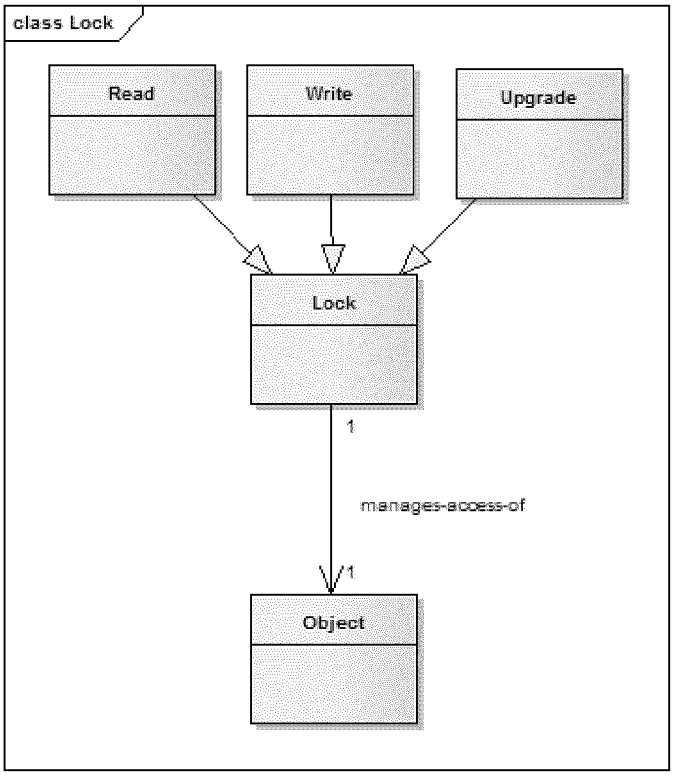
Concept	Superconcept	Definition	Subontology
Operations metadata	Meta object, Scope	Operations model the behavior that application objects support. They maintain a signature list of Parameters and refer to a result type. Operations may raise Exceptions.	Metadata.Metaobject
Exceptions metadata	Meta object	Operations may raise Exceptions and thereby return a different set of results. Exceptions refer to a Structure that defines their results and keep track of the Operations that may raise them.	Metadata.Metaobject
Properties metadata	Meta object	Properties form an abstract class over the Attribute and Relationship meta objects that define the abstract state of an application object. They have an associated type.	Metadata.Metaobject
Attributes metadata	Properties metadata	Attributes are properties that maintain simple abstract state. They may be read-only, in which case there is no associated accessor for changing their values.	Metadata.Metaobject
Relationships metadata	Properties metadata	Relationships model bilateral object references between participating objects. In use, two relationship meta objects are required to represent each traversal direction of the relationship. Operations are defined implicitly to form and drop the relationship, as well as accessor operations for manipulating its traversals.	Metadata.Metaobject
TypeDefinitions	Meta object	TypeDefinitions are meta objects that define new names, or aliases, for the types to which they refer. They participate in a number of relationships with the other meta objects that use them. These relationships allow	Metadata.Metaobject

		Types to be easily administered within the repository and help to ensure the referential integrity of the repository as a whole.	
Interfaces metadata	TypeDefinitions, DefiningScope	Interfaces are the most important types in the repository. Interfaces define the abstract behavior of application objects and contain operations for creating and removing Attributes, Relationships, and Operations within themselves in addition to the operations inherited from DefiningScope. Interfaces are linked in a multiple-inheritance graph with other Inheritance objects by two relationships, inherits and derives. They may contain most kinds of MetaObjects, except Modules and Interfaces.	Metadata.Metaobject
Classes metadata	Interfaces metadata	Classes are a subtype of Interface whose properties define the abstract state of objects stored in an ODMS. Classes are linked in a single inheritance hierarchy whereby state and behavior are inherited from an extender class. Classes may define keys and extents over their instances.	Metadata.Metaobject
Collections metadata	TypeDefinitions	Collections are types that aggregate variable numbers of elements of a single subtype and provide different ordering, accessing, and comparison behaviors. The maximum size of the collection may be specified by a constant or constant expression. If unspecified, this relationship will be bound to the literal 0.	Metadata.Metaobject
Constructed types	TypeDefinitions	Some types contain named elements that themselves refer to other types and are said to	Metadata.Metaobject

		be constructed from those types.	

Name	Concepts	Description	Subontology
is-a	Operation metadata - Meta object	Operation metadata is a meta object.	Metadata.Metaobject
is-a	Operation metadata - Scope	Operation metadata is a scope.	Metadata.Metaobject
is-a	Exception metadata - Meta object	Exception metadata is a meta object.	Metadata.Metaobject
is-a	Property metadata - Meta object	Property metadata is a meta object.	Metadata.Metaobject
is-a	Attribute metadata - Property metadata	Attribute metadata is a Property metadata.	Metadata.Metaobject
is-a	Relationship metadata - Property metadata	Relationship metadata is a Property metadata.	Metadata.Metaobject
is-a	TypeDefintion - Meta object	TypeDefinitions are meta objects that define new names, or aliases, for the types to which they refer.	Metadata.Metaobject
is-a	Interface metadata - TypeDefinition	Interface metadata is a TypeDefinition.	Metadata.Metaobject
is-a	Class metadata - TypeDefinition	Class metadata is a TypeDefinition.	Metadata.Metaobject
is-a	Collection metadata - TypeDefinition	Collection metadata is a TypeDefinition.	Metadata.Metaobject
is-a	Constructed type - TypeDefinition	Constructed type is a TypeDefinition.	Metadata.Metaobject
assigns-a-name	Specifier - TypeDefiniton	Specifiers are used to assign a name to a type in certain contexts.	Metadata.Metaobject

A zár részontológia



Az ODMG objektum-modell a konkurencia-vezérléshez hagyományos, zár-alapú megközelítést alkalmaz. A zár típusa: írási, olvasási és felminősítő. Az olvasási zár megosztott, míg az írási kizárólagos hozzáférést biztosít. A felminősítő zár a holtpont elkerülését szolgálja.

Concept	Superconcept	Definition	Subontology
Lock	CONCEPT	The ODMG Object Model uses a conventional lock-based approach to concurrency control. This approach provides a mechanism for enforcing shared or exclusive access to objects.	Lock
Read	Lock	Read locks allow shared access to an object.	Lock
Write	Lock	Write locks indicate exclusive access to an object. Readers of a particular object do not conflict with other readers, but writers conflict with both readers and writers.	Lock

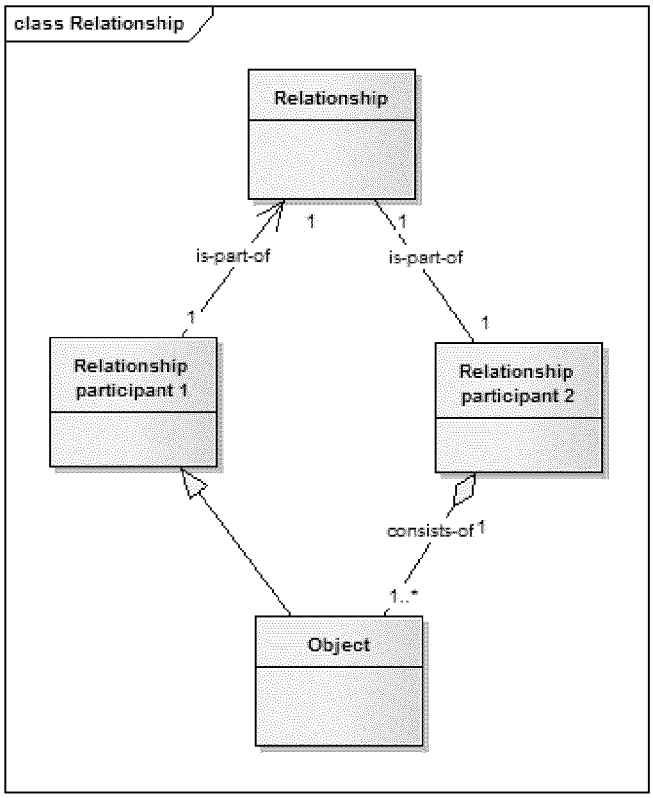
Upgrade	Lock	Upgrade locks are used to prevent a form of deadlock that occurs when two processes both obtain read locks on an object and then attempt to obtain write locks on that same object. Upgrade locks are compatible with read locks, but conflict with upgrade and write locks.	Lock

Name	Concepts	Description	Subontology
is-a	Read - Lock	Read is a lock.	Lock
is-a	Write - Lock	Write is a lock.	Lock
is-a	Upgrade - Lock	Upgrade is a lock.	Lock
manages-access-of	Lock - Object	The ODMG Object Model uses a conventional lock-based approach to concurrency control. This approach provides a mechanism for enforcing shared or exclusive access to objects.	Lock

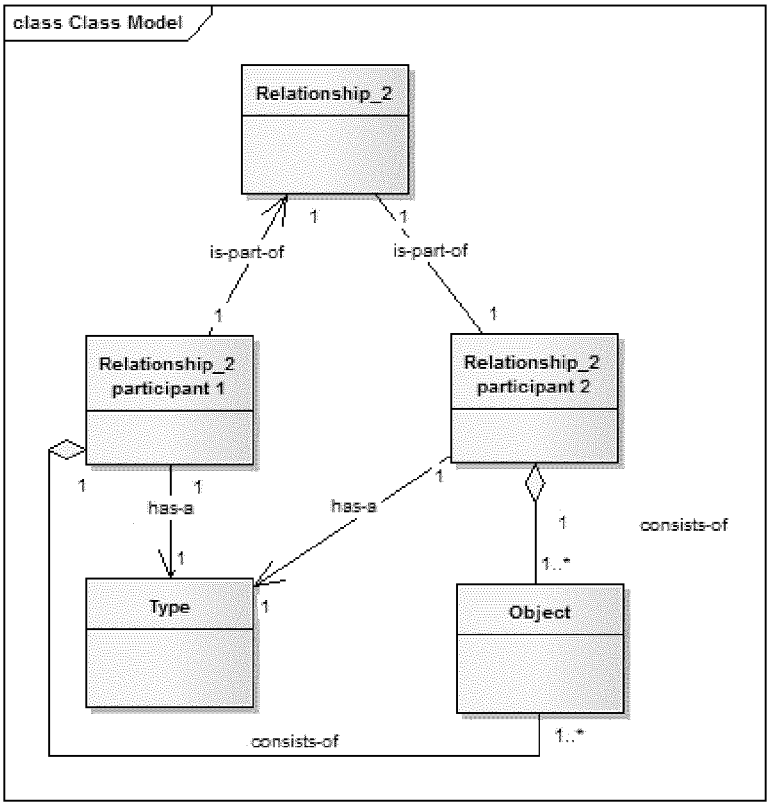
A szabvány inkonzisztenciái és nem-teljessége

Az ontológia készítése közben a szabvány számos hibáját fedeztem fel. Elsőként azon hibákat sorolom fel, melyektől az objektum-modell nem teljes: a tárolási osztály (storage domain) nem definiált a szabványban, de többször is hivatkozik rá. A tulajdonság és az attribútum definíciója is hibás: az attribútum a tulajdonság egy fajtája, de a definíciójuk mégis megegyezik.

A fenti hibák mellett inkonzisztenciák is találhatók a szabványban: a kapcsolat definíciója ilyen. A szabvány ([1], 1. oldal), az alábbi módon határozza meg ezt a fogalmat: „a tulajdonságok attribútumok, vagy egy objektum és más objektumok közti kapcsolat lehet."



A második definíciót a 30. oldalon találjuk: „A kapcsolatokat típusok között adjuk meg. Az ODMG objektum-modellje csak a bináris kapcsolatot engedi meg, azaz két típus között. A modell nem támogatja a kettőnél több szereplős kapcsolatokat."



A két definíciót áttekintve észrevehetjük az inkonzisztenciát: a kapcsolat először objektumok, majd típusok között definiált. Az első definíció emellett lehetővé teszi a többszereplős kapcsolatokat, míg a második explicit módon tiltja azt.

Újabb inkonzisztenciát találhatunk az interfész definíciója kapcsán: az objektum-modell szerint az „interfész megadása az objektum-típus absztrakt viselkedésmódjának megadása". A viselkedésmód definiálása az „objektumokon végrehajtható műveletek halmazának megadásával történik". Ezzel szemben a 29. oldalon a következő szerepel: „az interfészek attribútum-deklarációi az egyedek absztrakt viselkedésmódját adják meg."

Konklúzió

A dolgozatomban bemutattam az általam készített ODMG 3.0 ontológiát. Az ontológia segítségével a szabvány hiányosságai, inkonzisztenciái feltárhatóak. Az alternatív reprezentációk segítségével egy, az eredeti szabványnál sokkal könnyebben áttekinthető modellt kaptunk. A részenkénti feldolgozás segítheti a részek behatóbb megismerését, amely

az eredeti szabványban sokkal nehezebb dolgot jelent az információk többen helyen való közlése miatt. Az ontológia segítségével az objektum-orientált adatbázist, illetve az objektum-relációs leképezést használó szoftverek minősége javítható.

A további célok között szerepel az ontológia segítségével egyéb inkonzisztenciák és hibák felkutatása szerepel, illetve – amint elkészült – az új szabvánnyal történő összevetése. A REFSENO formalizmust is érdemes lenne bővíteni véleményem szerint, hiszen jelenlegi formájában a szinonimákat nem kezeli, amely fogalomrendszer esetében fontos lehet. Rövidtávú cél a dolgozat angol nyelvre fordítása a mostani kétnyelvű változat helyett.

Hosszútávon érdemes lehet kipróbálni a szoftverfejlesztési ontológiák másik fajtáját, a szoftverfejlesztési folyamatokban használt ontológiákat is. Elképezhető egy olyan projekt is, amelyben a koncepcionális modell helyett ontológiában tárolnánk a tárgyterülethez kapcsolódó ismeretek, és eközben a fejlesztés folyamán folyamati ontológiákat is használnánk. Érdemes lenne a két dolog összekapcsolását megvizsgálni.

Hivatkozások

1. R. G. G. Cattel, D. K. Barry (ed).: The Object Data Standard: ODMG 3.0, 2001, Morgan Kaufmann Publishers

2. C. Calero, F. Ruiz, M. Piattini (Eds.): Ontologies for Software Engineering and Software Technology, Springer, 2006

3. Tautz, C., Von Wangenheim, C.: REFSENO: A Representation Formalism for Software Engineering Ontologies. Fraunhofer IESE-Report No., 015.98/E, version 1.1, October 20, 1998.

4. Dr. Waralak V. Siricharoen: Ontologies and Object models in Object Oriented Software Engineering, IAENG International Journal of Computer Science, 33:1, IJCS_33_1_4, 2007

5. Wongthongtham, P., Chang, E., Dillon, T.S., Sommerville, I. (2008) 'Development of a Software Engineering Ontology for Multi-site Software Development', IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING, USA, Accepted on Sept., 19, 2008

6. SEOntology, http://www.seontology.org

7. W. Hesse: Engineers discovering the "real world" – From Model-driven to Ontology-based Software Engineering. Proc. 7th Int. Conf. on Information Systems Technology and its Applications ISTA2008; Klagenfurt/Austria (Invited Talk); published in Information Systems and e-Business Technologies, Volume 5(2), Springer Berlin Heidelberg, pp. 136-147; DOI 10.1007/978-3-540-78942-0

8. W. Hesse: Ontologies in the Software Engineering process. in: R. Lenz et al. (Hrsg.): EAI 2005 - Tagungsband Workshop on Enterprise Application Integration, GITO-Verlag

Berlin 2005

9. UNISCON 2008, http://www.uniscon2008.org/index.php?option=com_content&task=view&id=51&Itemid=74

10. Gruber TR (1993b) Toward principles for the design of ontologies used for knowledge sharing. In: Guarino N, Poli R (eds) International Workshop on Formal Ontology in Conceptual Analysis and Knowledge Representation. Padova, Italy. (Formal Ontology in Conceptual Analysis and Knowledge Representation) Kluwer Academic Publishers, Deventer, The Netherlands. http://citeseer.nj.nec.com/gruber93toward.html

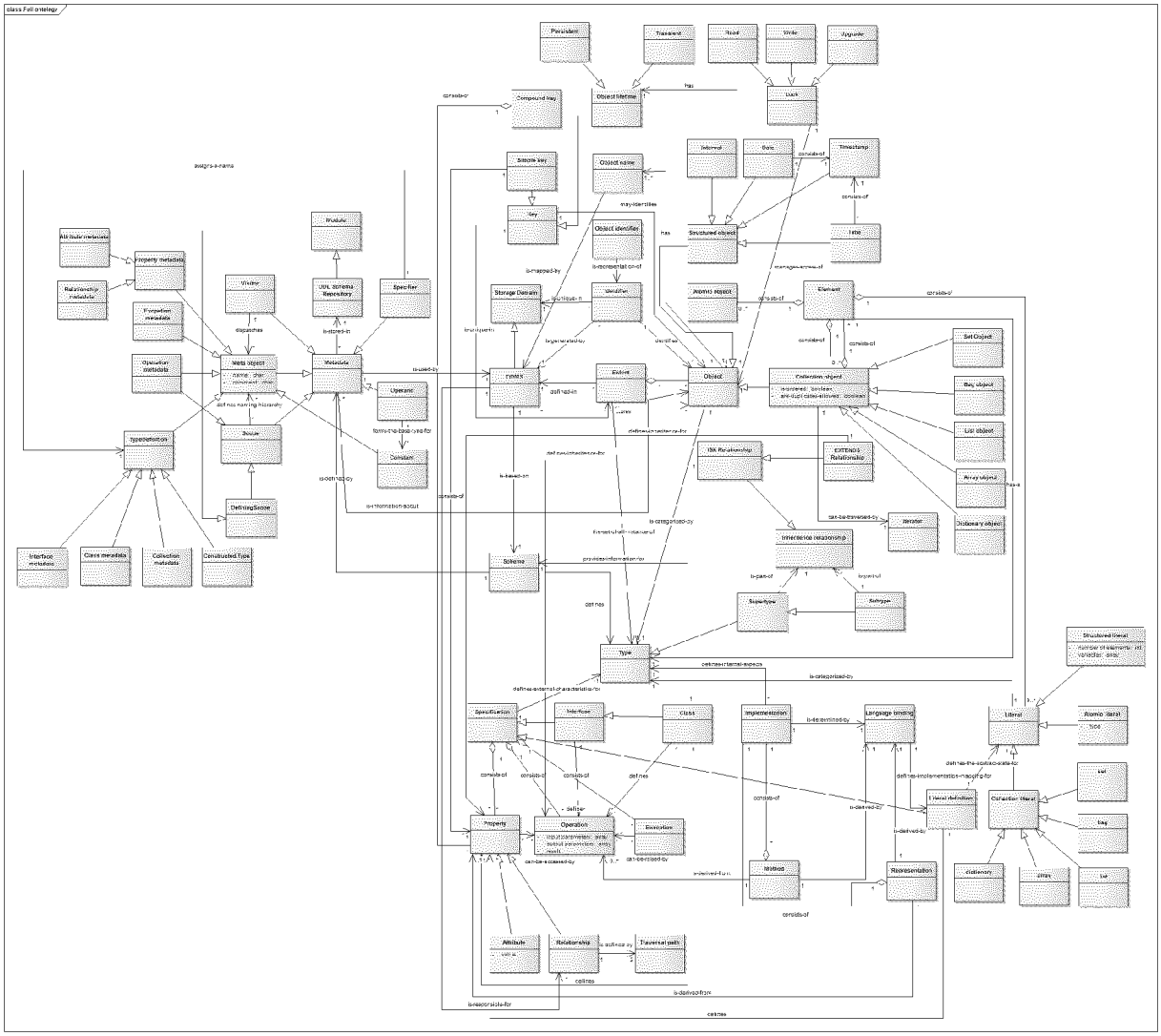
11. Alarcos, http://alarcos.inf-cr.uclm.es/defaultEng.aspx

12. Calero, C., Ruiz, F., Baroni, A., Brito e Abreu, F. and Piattini, M: An ontological approach to describe the SQL: 2003 object-relational features. Computer Standards & Interfaces. Elsevier. ScienceDirect. In press Available online 2 December 2005.

13. Francisco Ruiz, Marcela Genero, Félix García, Mario Piattini and Coral Calero: A proposal of a Software Measurement Ontology, www.frcu.utn.edu.ar/deptos/depto_3/32JAIIO/asse/asse_02.pdf

14. Next-Generation Object Database Standardization, http://www.odbms.org/download/033.01%20Card%20Next-Generation%20Object%20Database%20Standardization%20September%202007.PDF

Mellékletek



Concept	Superconcept	Definition	Subontology
Literal	CONCEPT	Literal is a basic modelling primitive. Literals can be categorized by their types.A literal has no identifier.	Literal
Atomic literal	Literal Types	Numbers and characters are examples of atomic literal types. Instances of these types are not explicitly created by applications, but rather implicitly exist.	Literal
Collection literal	Literal Types	Analogues to the collection objects.	Literal
Structured literal	Literal Types	A structured literal, or simply structure, has a fixed number of elements, each of which has a variable name and can contain either a literal value or an object.	Literal
set	Collection literal	Analogous to Set object.	Literal
bag	Collection literal	Analogous to Bag object.	Literal
list	Collection literal	Analogous to List object.	Literal
array	Collection literal	Analogous to Array object.	Literal
dictionary	Collection literal	Analogous to Dictionary object.	Literal
Lock	CONCEPT	The ODMG Object Model uses a conventional lock-based approach to concurrency control. This approach provides a mechanism for enforcing shared or exclusive access to objects.	Lock
Read	Lock	Read locks allow shared access to an object.	Lock
Write	Lock	Write locks indicate exclusive access to an object. Readers of a particular object do not conflict with other readers, but writers conflict with both readers and writers.	Lock
Upgrade	Lock	Upgrade locks are used to prevent a form of deadlock that occurs when two processes both obtain read locks on an object and then attempt to obtain write locks on that same object. Upgrade locks are compatible with read locks, but conflict with upgrade and	Lock

		write locks.	
ODL Schema Repository	Modul	Metadata is stored in an ODL Schema Repository, which is also accessible to tools and applications using the same operations that apply to user-defined types.	Metadata
Meta object	Object	Subclasses of MetaObject interface.	Metadata
Scope	CONCEPT	Scopes define a naming hierarchy for the meta objects in the repository.	Metadata
DefiningScope	Scope	DefiningScopes are Scopes that contain other meta object definitions using their defines relationship and that have operations for creating, adding, and removing meta objects within themselves.	Metadata
Visitor	CONCEPT	Visitors provide a convenient "double dispatch" mechanism for traversing the meta objects in the repository.	Metadata
Modul	Meta object, DefiningScope	Modules are DefiningScopes that define operations for creating modules and interfaces within themselves.	Metadata
Constants	Meta object	Constants provide a mechanism for statically associating values with names in the repository. The value is defined by an Operand subclass that is either a literal value (Literal), a reference to another Constant (ConstOperand), or the value of a constant expression (Expression). Each constant has an associated type and keeps track of the other ConstOperands that refer to it in the repository. The value operation allows the constant's actual value to be computed at any time.	Metadata
Specifier	Interface	Specifiers are used to assign a	Metadata

		name to a type in certain contexts.	
Operand	Interface	Operands form the base type for all constant values in the repository.	Metadata
Operations metadata	Meta object, Scope	Operations model the behavior that application objects support. They maintain a signature list of Parameters and refer to a result type. Operations may raise Exceptions.	Metadata.Metaobject
Exceptions metadata	Meta object	Operations may raise Exceptions and thereby return a different set of results. Exceptions refer to a Structure that defines their results and keep track of the Operations that may raise them.	Metadata.Metaobject
Properties metadata	Meta object	Properties form an abstract class over the Attribute and Relationship meta objects that define the abstract state of an application object. They have an associated type.	Metadata.Metaobject
Attributes metadata	Properties metadata	Attributes are properties that maintain simple abstract state. They may be read-only, in which case there is no associated accessor for changing their values.	Metadata.Metaobject
Relationships metadata	Properties metadata	Relationships model bilateral object references between participating objects. In use, two relationship meta objects are required to represent each traversal direction of the relationship. Operations are defined implicitly to form and drop the relationship, as well as accessor operations for manipulating its traversals.	Metadata.Metaobject
TypeDefinitions	Meta object	TypeDefinitions are meta objects that define new names, or aliases, for the types to which they refer. They participate in a number of relationships with the other meta objects that use them. These relationships allow Types to be easily administered within the repository and help to	Metadata.Metaobject

		ensure the referential integrity of the repository as a whole.	
Interfaces metadata	TypeDefinitions, DefiningScope	Interfaces are the most important types in the repository. Interfaces define the abstract behavior of application objects and contain operations for creating and removing Attributes, Relationships, and Operations within themselves in addition to the operations inherited from DefiningScope. Interfaces are linked in a multiple-inheritance graph with other Inheritance objects by two relationships, inherits and derives. They may contain most kinds of MetaObjects, except Modules and Interfaces.	Metadata.Metaobject
Classes metadata	Interfaces metadata	Classes are a subtype of Interface whose properties define the abstract state of objects stored in an ODMS. Classes are linked in a single inheritance hierarchy whereby state and behavior are inherited from an extender class. Classes may define keys and extents over their instances.	Metadata.Metaobject
Collections metadata	TypeDefinitions	Collections are types that aggregate variable numbers of elements of a single subtype and provide different ordering, accessing, and comparison behaviors. The maximum size of the collection may be specified by a constant or constant expression. If unspecified, this relationship will be bound to the literal 0.	Metadata.Metaobject
Constructed types	TypeDefinitions	Some types contain named elements that themselves refer to other types and are said to be constructed from those types.	Metadata.Metaobject

Object	CONCEPT	Object is a basic modelling primitive. Each object has a unique identifier, a state and a behavior. An object is sometimes referred to as an instance of its type.	Object
Identifier	CONCEPT	Identifies an object in a storage domain.	Object
Object identifier	Representation	The representation of the identity of an object is referred to as its object identifier. An object retains the same object identifier for its entire lifetime. Object identifiers are generated by the ODMS.	Object
Object name	CONCEPT	In addition to being assigned an object identifier by the ODMS, an object may be given one or more names that are meaningful to the programmer or end user.	Object
Object lifetime	CONCEPT	The lifetime of an object determines how the memory and storage allocated to the object are managed. The lifetime of an object is specified at the time the object is created.	Object
Persistent	Object lifetime	An object whose lifetime is persistent is allocated memory and storage managed by the ODMS runtime system.	Object
Transient	Object lifetime	An object whose lifetime is transient is allocated memory that is managed by the programming language runtime system.	Object
Key	CONCEPT	Keys are some property or set of properties which can identifiy the individual objects by the values they carry in some cases.	Object.Key
Simple key	Key	A simple key is a key consists of a single property.	Object.Key
Compund key	Key	A compound key is a key consists of a set of properties.	Object.Key
Atomic object	Object	Atomic object is an object which is user-defined.	Object.Types

Collection object	Object	In the ODMG Object Model, instances of collection objects are composed of distinct elements, each of which can be an instance of an atomic type, another collection, or a literal type. All the elements of the collection must be of the same type.	Object.Types
Structured object	Object	All structured objects support the Object ODL interface.	Object.Types
Iterator	CONCEPT	An iterator is a mechanism for accessing the elements of a Collection object, can be created to traverse a collection.	Object.Types.Collection
Set object	Collection objects	A Set object is an unordered collection of elements, with no duplicates allowed.	Object.Types.Collection
Bag object	Collection objects	A Bag object is an unordered collection of elements that may contain duplicates.	Object.Types.Collection
List object	Collection objects	A List object is an ordered collection of elements.	Object.Types.Collection
Array object	Collection objects	An Array object is a dynamically sized, ordered collection of elements that can be located by position.	Object.Types.Collection
Dictionary object	Collection objects	A Dictionary object is an unordered sequence of key-value pairs with no duplicate keys.	Object.Types.Collection
Element	CONCEPT	For modelling reasons: element of a collection. Can be an instance of an atomic type, another collection, or a literal type. All the elements of the collection must be of the same type.	Object.Types.Collection
Date	Structured objects	A structured object defined by the DateFactory interface and the Date class.	Object.Types.Structured
Interval	Structured objects	Intervals represent a duration of time and are used to perform some operations on Time and Timestamp objects.	Object.Types.Structured
Time	Structured objects	Times denote specific world times, which are internally stored in Greenwich Mean Time (GMT).	Object.Types.Structured
Timestamp	Structured	Timestamps consist of an	Object.Types.Structured

	objects	encapsulated Date and Time.	
ODMS	CONCEPT	ODMS is Object Data Management System. An ODMS stores objects, enabling them to be shared by multiple users and applications.	ODMS
Schema	CONCEPT	An ODMS is based on a schema that is defined in ODL and contains instances of the types defined by its schema.	ODMS
Extent	CONCEPT	The extent of a type is the set of all instances of the type within a particular ODMS.	ODMS
Storage domain	CONCEPT		ODMS
Metadata	CONCEPT	Metadata is descriptive information about persistent objects that defines the schema of an ODMS. Metadata is used by the ODMS to define the structure of its object storage, and at runtime, guides access to the ODMS's persistent objects.	ODMS
Type	CONCEPT	Types categorizes objects and literals. All elements of a given type have a common range of states (i.e., the same set of properties) and common behavior (i.e., the same set of defined operations). A type has an external specification and one or more implementations.	Type
Specification	CONCEPT	The specification defines the external characteristics of the type. These are the aspects that are visible to users of the type: the operations that can be invoked on its instances, the properties, or state variables, whose values can be accessed, and any exceptions that can be raised by its operations.	Type
Implementation	CONCEPT	A type's implementation defines the internal aspects of the objects of the type: the implementation of the type's operations and other internal details. An implementation of an object type consists of a representation and a	Type

		set of methods.	
Language binding	CONCEPT	The implementation of a type is determined by a language binding.	Type.Implementation
Representation	CONCEPT	The representation is a data structure that is derived from the type's abstract state by a language binding: For each property contained in the abstract state there is an instance variable of an appropriate type defined.	Type.Implementation
Method	CONCEPT	The methods are procedure bodies that are derived from the type's abstract behavior by the language binding: For each of the operations defined in the type's abstract behavior a method is defined. This method implements the externally visible behavior of an object type. A method might read or modify the representation of an object's state or invoke operations defined on other objects. There can also be methods in an implementation that have no direct counterpart to the operations in the type's specification.	Type.Implementation
Inheritence relationship	CONCEPT	A relationship between two types, when one type (the subtype) inherits some characteristics of the other (the supertype).	Type.Inheritence
ISA relationship	Inheritence relationship	Defines the inheritence of behavior between two types.	Type.Inheritence
EXTENDS relationship	Inheritence relationship	Defines the inheritence of behavior and state between two types. Applies only to object types.	Type.Inheritence
Supertype	Type	Is a part of the inheritence relationship and is a type.	Type.Inheritence
Subtype	Supertype	Is a part of the inheritence relationship and is a type same as it supertype's with additional abstract state or behavior.	Type.Inheritence
Property	CONCEPT	The values carried by a set of	Type.Specification

		properties defines the state of an object.	
Attribute	Property	Defines the abstract state of an object.	Type.Specification
Relationship	Property	A relationship is defined between two types, each of which must have instances that are referenceable by object identifiers. The ODMS is responsible for maintaining the referential integrity of relationships. The implementation of relationships is encapsulated by public operations that form and drop members from the relationship, plus public operations on the relationship target classes to provide access and to manage the required referential integrity constraints.	Type.Specification
Operation	CONCEPT	The behavior of an object is defined by the set of operations that can be executed on or by the object. An operation has a name, which can be overloaded. Operations may have a list of input and output parameters, each with a specified type. Each operation may also return a typed result. Some operations may return no value. An operation may have side effects. The Object Model assumes sequential execution of operations.	Type.Specification
Exception	CONCEPT	Operations can raise exceptions, and exceptions can communicate exception results. Mappings for exceptions are defined by each language binding.	Type.Specification
Interface	Specification	An interface definition is a specification that defines only the abstract behavior of an object type.	Type.Specification
Class	Specification	A class definition is a specification that defines the abstract behavior and abstract	Type.Specification

		state of an object type. A class is an extended interface with information for ODMS schema definition.	
Literal definition	Specification	A literal definition defines only the abstract state of a literal type.	Type.Specification
Traversal path	CONCEPT	A relationship is defined explicitly by declaration of traversal paths that enable applications to use the logical connections between the objects participating in the relationship. Traversal paths are declared in pairs, one for each direction of traversal of the relationship.	Type.Specification

Concept	Attribute	Description	Cardinality	Subontology
Attribute	value	Attribute has a value.	1	Type.Specification
Operation	input parameter	Operations may have a list of input and output parameters, each with a specified type.	*	Type.Specification
	output paramater	Operations may have a list of input and output parameters, each with a specified type.	*	
	result	Each operation may also return a typed result. Some operations may return no value.	1	
Atomic literal	type	The ODMG Object Model supports the following types of atomic literals: • long • long long • short • unsigned long • unsigned short • float • double • boolean • octet • char (character) • string • enum (enumeration)	1	Literal
Structured literal	number of elements	A structured literal, or simply structure, has a fixed number of elements, each of which has a variable name and can contain either a literal value or an object.	1	Literal
	variables	A structured literal, or simply structure, has a fixed number of elements, each of which has a variable name and can contain either a literal value or an object.	*	
Meta object	name	Meta objects have names.	1	Metadata
	comment	Meta objects stores comments.	1	
Collection				Object.Types
	is-ordered	Declares the orderedness of	1	

		the object.		
object	are-duplicates-allowed	Declares whether the duplicates allowed.	1	

Name	Concepts	Description	Subontology
is-a	Atomic literal - Literal	Atomic literal is a literal.	Literal
is-a	Collection literal - Literal	Collection literal is a literal.	Literal
is-a	set - Collection literal	set is a collection literal.	Literal
is-a	bag - Collection literal	bag is a collection literal.	Literal
is-a	list - Collection literal	list is a collection literal.	Literal
is-a	array - Collection literal	array is a collection literal.	Literal
is-a	dictionary - Collection literal	dictionary is a collection literal.	Literal
is-a	Structured literal - Literal	Structured literal is a literal.	Literal
is-a	Read - Lock	Read is a lock.	Lock
is-a	Write - Lock	Write is a lock.	Lock
is-a	Upgrade - Lock	Upgrade is a lock.	Lock
manages-access-of	Lock - Object	The ODMG Object Model uses a conventional lock-based approach to concurrency control. This approach provides a mechanism for enforcing shared or exclusive access to objects.	Lock
is-stored-in	Metadata - ODL Schema Repository	Metadata is stored in an ODL Schema Repository.	Metadata
is-a	Meta object - Metadata	Meta object is a metadata.	Metadata
is-a	Scope - Metadata	Scope is a metadata.	Metadata
is-a	Visitor - Metadata	Visitor is a metadata.	Metadata
is-a	Operand - Metadata	Operand is a metadata.	Metadata
is-a	Specifier - Metadata	Specifier is a metadata.	Metadata
defines-naming-hierarchy-for	Scope - Meta object	Scopes define a naming hierarchy for the meta objects in the repository.	Metadata

is-a	DefiningScope - Scope	DefiningScopes are Scopes that contain other meta object definitions using their defines relationship and that have operations for creating, adding, and removing meta objects within themselves.	Metadata
dispatches	Visitor - Meta object	Visitors provide a convenient "double dispatch" mechanism for traversing the meta objects in the repository.	Metadata
is-a	Module - DefiningScope	Modules and the Schema Repository itself, which is a specialized module, are Defining-Scopes that define operations for creating modules and interfaces within themselves.	Metadata
is-a	ODL Schema Repository - Module	Modules and the Schema Repository itself, which is a specialized module, are Defining-Scopes that define operations for creating modules and interfaces within themselves.	Metadata
is-a	Constant - Meta object	Constant is a meta object.	Metadata
forms-the-base-type-for	Operand - Constant	Operands forms the base type for constants.	Metadata
is-a	Operation metadata - Meta object	Operation metadata is a meta object.	Metadata.Metaobject
is-a	Operation metadata - Scope	Operation metadata is a scope.	Metadata.Metaobject
is-a	Exception metadata - Meta object	Exception metadata is a meta object.	Metadata.Metaobject
is-a	Property metadata - Meta object	Property metadata is a meta object.	Metadata.Metaobject
is-a	Attribute metadata - Property metadata	Attribute metadata is a Property metadata.	Metadata.Metaobject

is-a	Relationship metadata - Property metadata	Relationship metadata is a Property metadata.	Metadata.Metaobject
is-a	TypeDefintion - Meta object	TypeDefinitions are meta objects that define new names, or aliases, for the types to which they refer.	Metadata.Metaobject
is-a	Interface metadata - TypeDefinition	Interface metadata is a TypeDefinition.	Metadata.Metaobject
is-a	Class metadata - TypeDefinition	Class metadata is a TypeDefinition.	Metadata.Metaobject
is-a	Collection metadata - TypeDefinition	Collection metadata is a TypeDefinition.	Metadata.Metaobject
is-a	Constructed type - TypeDefinition	Constructed type is a TypeDefinition.	Metadata.Metaobject
assigns-a-name	Specifier - TypeDefiniton	Specifiers are used to assign a name to a type in certain contexts.	Metadata.Metaobject
stores	ODMS - Object	ODMS stores objects	Object
identifies	Identifier - Object	All identifiers of objects in an ODMS are unique, relative to each other.	Object
is-generated-by	Identifier - ODMS	Object identifiers are generated by the ODMS.	Object
is-unique-in	Identifier - Storage domain	Because all objects have identifiers, an object can always be distinguished from all other objects within its storage domain.	Object
is-representation-of	Object identifier - Identifier	The representation of the identity of an object is referred to as its object identifier.	Object
has	Object - Object name	Object has one or more object name	Object
has	Object - Object lifetime	Object has a lifetime.	Object
is-a	Persistent - Object lifetime	Persistent is a lifetime.	Object
is-a	Transient - Object lifetime	Transient is a lifetime.	Object
mapped-by	Object name - ODMS	The ODMS provides a function that it uses to map from an object name to an object.	Object

may-identifies	Key - Object	In some cases, the individual instances of a type can be uniquely identified by the values they carry for some property or set of properties. These identifying properties are called keys.	Object.Key
is-unique-in	Key - Extent	The scope of uniqueness is the extent of the type; thus, a type must have an extent to have a key.	Object.Key
is-a	Simple key - Key	Simple key is a key.	Object.Key
consists-of	Simple key - Property	A simple key consists of a single property.	Object.Key
is-a	Compound key - Key	Compound key is a key.	Object.Key
consists-of	Compound key - Property	A compund key consists of a set of properties.	Object.Key
is-a	Atomic object - Object	An atomic object type is an user-defined object.	Object.Types
is-a	Collection object - Object	Collection object is an object.	Object.Types
is-a	Structured object - Object	Structured object is an object.	Object.Types
consists-of	Collection object - Element	In the ODMG Object Model, instances of collection objects are composed of distinct elements, each of which can be an instance of an atomic type, another collection, or a literal type.	Object.Types.Collection
consists-of	Element - Atomic object	In the ODMG Object Model, instances of collection objects are composed of distinct elements, each of which can be an instance of an atomic type, another collection, or a literal type.	Object.Types.Collection

consists-of	Element - Collection object	In the ODMG Object Model, instances of collection objects are composed of distinct elements, each of which can be an instance of an atomic type, another collection, or a literal type.	Object.Types.Collection
consists-of	Element - Literal	In the ODMG Object Model, instances of collection objects are composed of distinct elements, each of which can be an instance of an atomic type, another collection, or a literal type.	Object.Types.Collection
is-a	Set object - Collection object	Set object is a collection object.	Object.Types.Collection
is-a	Bag object - Collection object	Bag object is a collection object.	Object.Types.Collection
is-a	List object - Collection object	List object is a collection object.	Object.Types.Collection
is-a	Array object - Collection object	Array object is a collection object.	Object.Types.Collection
is-a	Dictionary object - Collection object	Dictionary object is a collection object.	Object.Types.Collection
can-be-traversed-by	Collection object - Iterator	An iterator is a mechanism for accessing the elements of a Collection object, can be created to traverse a collection.	Object.Types.Collection
is-a	Date - Structured object	Date is a structured object.	Object.Types.Structured
is-a	Interval - Structured object	Interval is a structured object.	Object.Types.Structured
is-a	Time - Structured object	Time is a structured object.	Object.Types.Structured
is-a	Timestamp - Structured object	Timestamp is a structured object.	Object.Types.Structured
consists-of	Timestamp - Date	Timestamps consist of an encapsulated Date and Time.	Object.Types.Structured
consists-of	Timestamp - Time	Timestamps consist of an encapsulated Date and Time.	Object.Types.Structured

defines	Schema - Type	An ODMS is based on a schema that is defined in ODL and contains instances of the types defined by its schema.	ODMS
is-based-on	ODMS - Schema	ODMS is based on a schema.	ODMS
the-set-of-all-instances-in	Extent - Type	The extent of a type is the set of all instances of the type within a particular ODMS.	ODMS
defined-in	Extent - ODMS	The extent of a type is the set of all instances of the type within a particular ODMS.	ODMS
consists-of	Extent - Object	The extent of a type is the set of all instances of the type within a particular ODMS.	ODMS
is-information-about	Metadata - Object	Metadata is descriptive information about persistent objects that defines the schema of an ODMS.	ODMS
is-defined-by	Schema - Metadata	Metadata is descriptive information about persistent objects that defines the schema of an ODMS.	ODMS
is-used-by	Metadata - ODMS	Metadata is used by the ODMS to define the structure of its object storage, and at runtime, guides access to the ODMS's persistent objects.	ODMS
is-a	ODMS - Storage domain	ODMS is a storage domain.	ODMS
defines-external-characteristics-for	Specification - Type	The specification defines the external characteristics of the type	Type
defines-internal-behavior-for	Implementation - Type	A type's implementation defines the internal aspects of the objects of the type.	Type
is-categorized-by	Object - Type	Objects can be categorized by their types.	Type
is-categorized-	Literal - Type	Literals can be categorized	Type

by		by their types.	
is-determined-by	Implementation - Language binding	The implementation of a type is determined by a language binding.	Type.Implementation
defines-implementation-mapping-for	Language binding - Literal defintion	Each language binding also defines an implementation mapping for literal types.	Type.Implementation
consists-of	Implementation - Representation	An implementation of an object type consists of a representation and a set of methods.	Type.Implementation
consists-of	Implementation - Method	An implementation of an object type consists of a representation and a set of methods.	Type.Implementation
derived-by	Representation - Language binding	Representation is derived by the language binding from the type's abstract state.	Type.Implementation
derived-by	Method - Language binding	Methods are derived by the language binding from the type's abstract behavior.	Type.Implementation
is-derived-from	Representation - Property	Representation is derived by the language binding from the type's abstract state.	Type.Implementation
is-derived-from	Method - Operation	Methods are derived by the language binding from the type's abstract behavior.	Type.Implementation
is-a	Supertype - Type	Supertype is a type.	Type.Inheritence
is-a	Subtype - Supertype	Subtype is a supertype.	Type.Inheritence
is-part-of	Supertype - Inheritence relationship	Supertype is a part of an inheritence relationship.	Type.Inheritence
is-part-of	Subtype - Inheritence relationship	Subtype is a part of an inheritence relationship.	Type.Inheritence
is-a	ISA relationship - Inheritence relationship	ISA relationship is an inheritence relationship.	Type.Inheritence
is-a	EXTENDS relationship - ISA relationship	EXTENDS relationship is an ISA relationship.	Type.Inheritence
defines-inheritence-for	ISA relationship - Operation	ISA defines the inheritance of behavior.	Type.Inheritence
defines-inheritence-for	EXTENDS relationship -	EXTENDS defines the inheritance of behavior and	Type.Inheritence

	Property	state.	
is-a	Interface - Specification	Interface is a specification.	Type.Specification
is-a	Class - Interface	Class is an interface.	Type.Specification
is-a	Literal definiton - Specification	Literal definiton is a specification.	Type.Specification
consists-of	Specification - Property	An external specification of a type consists of an implementation-independent, abstract description of the operations, exceptions, and properties that are visible to users of the type.	Type.Specification
consists-of	Specification - Operation	An external specification of a type consists of an implementation-independent, abstract description of the operations, exceptions, and properties that are visible to users of the type.	Type.Specification
consists-of	Specification - Exception	An external specification of a type consists of an implementation-independent, abstract description of the operations, exceptions, and properties that are visible to users of the type.	Type.Specification
defines	Interface - Property	Interface defines abstract state.	Type.Specification
defines	Class - Operation	Class defines abstract behavior.	Type.Specification
defines	Literal definition - Property	Literal definition defines abstract state.	Type.Specification
can-be-accessed-by	Property - Operation	Properties can be accessed by operations.	Type.Specification
can-be-raised-by	Exception - Operation	Exceptions can be raised by operations.	Type.Specification
is-a	Property - Attribute	Attribute is a Property.	Type.Specification
is-a	Relationship - Attribute	Relationship is a Property.	Type.Specification

is-defined-by	Relationship - Traversal Path	A relationship is defined explicitly by declaration of traversal paths that enable applications to use the logical connections between the objects participating in the relationship. Traversal paths are declared in pairs, one for each direction of traversal of the relationship.	Type.Specification
is-responsible-for	ODMS - Relationship	The ODMS is responsible for maintaining the referential integrity of relationships.	Type.Specification
defines-the-abstract-state-for	Literal definition - Literal	A literal definition defines only the abstract state of a literal type.	Type.Specification