Ontologies and Rules for Enterprise Modeling and Simulation

Gerd Wagner
Institute of Informatics
Brandenburg University of Technology
Cottbus, Germany
G.Wagner@tu-cottbus.de

***Abstract*—We propose to model an enterprise as an institutional agent with organizational units and human actors as subagents that participate in zero or more business processes involving other subagents of the enterprise and other agents, which are possibly affiliated with other organizations. Our approach, which unifies state structure and behavior modeling, leads to a more holistic model of an enterprise, and its multitude of business processes, compared to traditional BPM approaches, such as Petri Nets and BPMN, which are exclusively focused on single business processes. We discuss the ontological foundations of our approach and show the superiority of our rule-based modeling and simulation language AORSL, which allows the operational modeling of entire business systems and their business processes.**

Keywords-ontologies; rules; enterprise modeling; agent-based simulation

I. INTRODUCTION)

In this paper we summarize some results we have attained in our effort to define the ontological foundations of enterprise modeling and simulation [6,7,10]. We consider an enterprise, and any other form of organization, as an *agent-based discrete event system*, or more precisely as an ***institutional agent***. In an ontology of agent-based discrete event systems, *rules* play the important role of transition functions advancing both the subjective state of agents and the objective state of the system under consideration.

In a series of publications [1-4] we have reported about our research project for developing a foundational ontology called "UFO" (standing for *Unified Foundational Ontology*) by employing theories from Formal Ontology, Cognitive Psychology, Linguistics, Philosophy of Language and Philosophical Logics. The core of UFO has been established through the development of an ontology of endurants in [5].

Since the development of UFO is an ongoing project, we use a simplified version of it, called *Essential Unified Foundational Ontology (eUFO)*, which restricts both the breadth and the depth of UFO, and simplifies its philosophical terminology, harmonizing it with common informatics terminology as much as possible.

Since our main concern are the ontological foundations of modeling languages for information system engineering and system simulation engineering, we derive from eUFO the foundational ontologies DESO and ABDESO, which are supposed to conceptualize the domains of (agent-based) information systems engineering and discrete event simulation engineering. However, since organizations, as institutional agents, come on top of simple agents, which come on top of objects and events, their ontological foundations are quite complex.

II. THE BASE LAYER OF EUFO

In this section, we briefly summarize the base layer of eUFO, called eUFO-0. The purpose of eUFO-0 is to lay the ground for categorizing the basic modeling concepts of data values, objects, events, sets, data types and object types.

eUFO-0 defines a number of basic ontological categories, as depicted in Fig. 1 below in the form of a conceptual UML class diagram, making a fundamental distinction between ***individuals***, which are things that exist in time and space in "the real world" and have a unique identity, and ***universals***, which are feature-based classifiers that classify, at any moment in time, a set of individuals with common features.

Well-known special cases of individuals are *objects* and *events*. Well-known special cases of universals are *object types* and *event types*. An example of an object is *the earth*; an example of a universal is the object type *planet*.

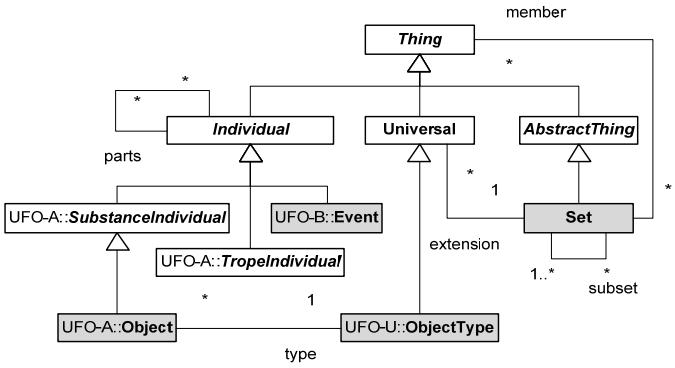

Figure 1. The base layer eUFO.-0 (common terms in grey)

We distinguish between three kinds of individuals: *substance individuals*, *trope individuals* and *events*. As opposed to substance individuals, trope individuals can only exist in other individuals, i.e., they are *existentially dependent* on other individuals. The distinction between substance individuals and events can be understood in terms of their relationship to time. Substance individuals are wholly present whenever they are present, i.e., they *are in time*. Events *happen in time*, they may have temporal parts.

Examples of substance individuals are: the person with name "Gerd Wagner", the moon, or an amount of sand. Examples of events are: today's rise of the sun, my

confirmation of an e-commerce purchase order through clicking the OK button, the sinking of the Titanic, or the Second World War. Examples of trope individuals are: the redness of John's T-shirt, Giancarlo's employment with UFES, or my daughter's belief in God.

In addition to individuals and universals, there are also ***abstract things***. As opposed to individuals and universals, abstract things are independent of space and time. A ***set*** is an abstract thing. The set of all instances of a universal is called its ***extension***.

In English, the term 'abstract' is overloaded, and one may consider the universal *planet* to be 'more abstract' than the individual *earth*. However, we consider things to be *abstract*, only, if they do not depend in any way on space and time. Thus, we do not consider universals, such as *planet*, to be abstract, since they depend, via their instances, on space and time.

Anything can be a member of a set. The *empty set* has no members and is a subset of all sets. The empty set is a *pure set*. Any set that has only pure sets as members is a *pure set* (in set theory, those things that are not sets are called 'urelements'). Thus, only pure sets are abstract, really.

The important distinction between object types and *data types* is explained by Fig. 2. While an object type is a universal, a ***data type*** is an abstract thing that is associated with two sets: its lexical space (a symbol set) and its value space, and with a symbol interpretation function mapping data literals from the lexical space to data values from the value space.

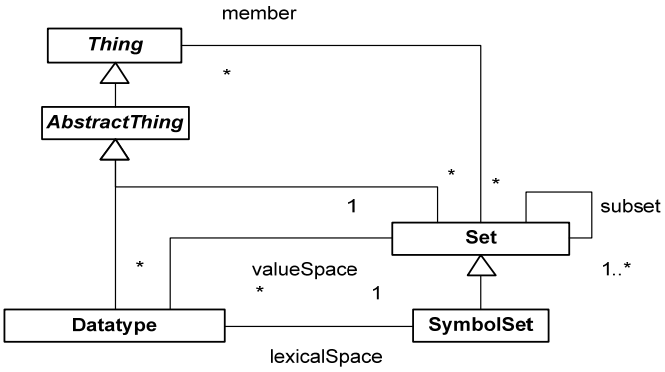

Figure 2. Data types as abstract things (eUFO-0).

The ontology of substance individuals and trope individuals forms the eUFO layer *A*, which is further discussed in section III, while the ontology of events forms the eUFO layer *B*, which is further discussed in section IV.

III. SUBSTANCE INDIVIDUALS AND TROPE INDIVIDUALS

Substance individuals possess (direct) spatio-temporal qualities and are founded on matter. We distinguish between two kinds of substance individuals: *physical* ***objects*** and ***quantities***, as depicted in Fig. 3.

Examples of physical objects, which we also call just "objects", include ordinary entities of everyday experience such as my car, Alan Turing, The Rolling Stones, or the North-Sea. Examples of quantities as individuals are these 5.75 liters of water or these 3 apples (notice that there is also the notion of a quantity as a type, which must not be confused with quantity individuals). In contrast with trope individuals, substance individuals do not inhere in anything and, as a consequence, they are (essentially) *existentially independent.*

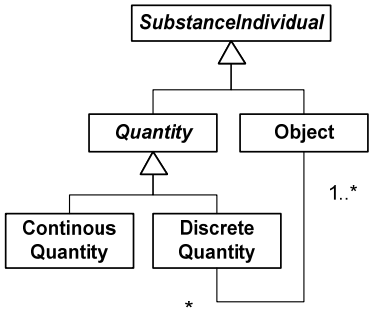

Figure 3. Substance individuals. (eUFO-A)

The redness of John's T-shirt is an example of a trope individual. It *inheres* in John's T-shirt, which is its bearer. Both John's T-shirt and the redness of John's T-shirt are individuals. However, they are individuals of very different natures. Trope individuals can only exist in other individuals, i.e., they are *existentially dependent* on other individuals in the way, for instance, the color and the weight of an apple *a* depend on *a*, the electric charge of a conductor *c* depends on *c*, or John's headache depends on John. In contrast, individuals such as John, the apple *a*, and the conductor *c* are not existentially dependent entities in this sense.

As depicted in Fig. 4, there are three basic kinds of trope individuals: ***qualities***, such as an individualized color, a temperature, or a weight, and ***modes***, such as a symptom, a skill, a belief, or an intention, are *intrinsic* trope individuals; whereas ***relators***, such as a kiss, a covalent bond, a medical treatment, a purchase order, or a social commitment, are *relational* trope individuals.

The special type of existential dependence relation that holds between a trope individual x and the individual y on which x depends is the relation of *inherence*. Existential dependence can also be used to differentiate intrinsic and relational trope individuals: qualities and modes are dependent on one single individual; relators depend on two or more individuals (their *bearers*), which they *mediate*.

There are two further kinds of trope individuals: ***attributions*** are associated with qualities, and ***material relationships*** are associated with relators. We discuss them in the following subsections.

A. Qualities and Attributions

According to the *non-migration* (or *non-transferability*) *principle*, it is not possible for a particular quality *q* to inhere in two different individuals *a* and *b*. This principle may seem counterintuitive. For example, if we have two particulars *a* (a red apple) and *b* (a red car), and two qualities r_1 (the particular redness of *a*) and r_2 (the particular redness of *b)*, we consider r_1 and r_2 to be different individuals.

What does it mean, then, to say that *a* and *b* have the same color? Sameness here cannot refer to strict (numerical) identity, but only to a qualitative one (i.e., equivalence in a certain respect). We thus distinguish between the color of a particular apple (as a *quality* of the apple) and the color *data value* that we associate with this quality in an ***attribution*** (with the help of an *attribute*). This data value is a member of the *value space* of the *data type* of the attribute associated with the corresponding *quality universal* (see Subsection 3.5).

As an example, consider the attribute *hairColor*, which is applicable to persons, and associated to a data type with a value space consisting of color names. Then, the triple <john, hairColor, "grey"> represents an attribution that makes the sentence "The hair color of John is grey" true. An attribution represents a *fact* that may be viewed as a "truth maker" for a corresponding sentence (an *attribution statement*) expressed in a suitable language.

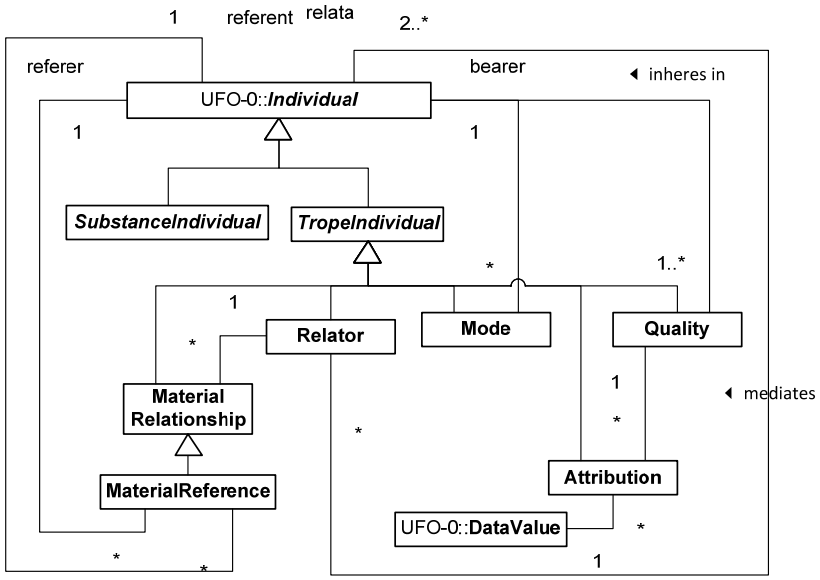

Figure 4. Trope individuals (eUFO-A).

B. Modes

Modes are, like qualities, intrinsic trope individuals. But unlike qualities, modes are not directly related to attributions.

C. Relators, Relationships and References

While a ***formal relationship***, such as [Brandenburg is part of Germany] holds directly, for a ***material relationship***, such as [Paul is being treated in the medical unit M], to exist, something else, which *mediates* the involved individuals (Paul and M), must exist. Such a mediating individual with the power of connecting individuals is called a ***relator***. For example, a medical treatment connects a patient with a medical unit; an enrollment connects a student with an educational institution; a covalent bond connects two atoms.

For any relator that mediates certain entities, there are one or more corresponding *material relationships* based on tuples constituted by these entities. Consider, for instance, the marriage of Paul and Marry. For this relator, there are three corresponding material relationships: [Paul is married to Marry], [Paul is husband of Marry] and [Mary is wife of Paul].

An important notion for the characterization of relators (and, hence, for the characterization of material relationships) is the notion of *foundation*, which can be viewed as a type of historical dependence in the way that, for example, an instance of being kissed is founded on an individual kiss. Suppose that John is married to Mary. In this case, we can assume that there is a particular relator m_1 of type marriage that mediates John and Mary. The foundation of this relator can be a wedding event or the signing of a contract between the involved parties. In other words, a certain event e_1, in which John and Mary participate, can create a particular marriage m_1 which existentially depends on John and Mary and which mediates them. The event e_1, in this case, is the foundation of the relator m_1. In general, relators are founded on events.

A ***material reference***, as an instance of a *material reference property*, is a binary material relationship between two individuals, the *referer* and the *referent*. As an example, consider the material reference property *wife*, which is a applicable to male persons. The triple <john, wife, mary> represents a reference that makes the sentence "John is married to Mary" true.

In a correspondence theory of truth (such as Tarski's semantics of predicate logic), attributions and material references, and other kinds of relationships, are considered as "truth makers" ("facts") that make corresponding sentences (*attribution statements* and *reference statements*) true.

Modeling and simulation languages normally include constructs for expressing attributions and references for specifying initial states.

IV. EVENTS

Events are individuals that may be composed of temporal parts. They *happen in time* in the sense that they may extend in time accumulating temporal parts. Examples of events are: the arrival of an incoming email message, a football game, a symphony execution, a birth of a mammal, the Second World War, or a particular business process. An event cannot exhibit change in time in a genuine sense since none of its temporal parts retain their identity through time.

Fig. 5 depicts the core fragment of the eUFO-B ontology of events. An event can be atomic or complex, depending on its mereological structure, i.e., while an *atomic event* has no parts, a *complex event* is an aggregation of at least two events (that can themselves be atomic or complex).

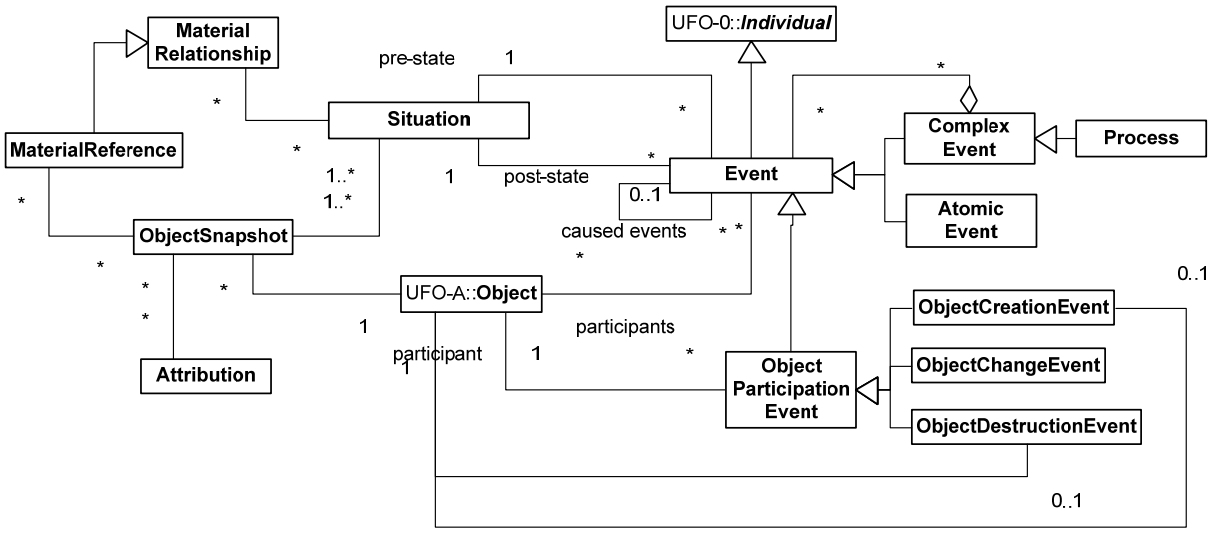

Figure 5. The ontology of events (eUFO-B).

Events are ontologically dependent entities in the sense that they existentially depend on their *participants* in order to exist. Take for instance the event *e*: the stabbing of Caesar by Brutus. In this event we have the participation of Caesar himself, of Brutus and of the knife. Each of these participations is itself an event (an *object participation event*), which existentially depends on a single object. Special cases of object participation events are *object creation*, *object change* and *object destruction* events.

Events may change the real world by changing the state of affairs from a *pre-state* situation to a *post-state* situation. Each situation is determined by a set of associated *object snapshots* and a set of associated material relationships holding between the involved objects, where an object snapshot is a set of attributions and references of a particular object.

Being atomic and being instantaneous are orthogonal notions in this framework, i.e., an atomic event can be time-extended and an instantaneous event can be composed of multiple (instantaneous) events.

All spatial properties of events are defined in terms of the spatial properties of their participants. In contrast, all temporal properties of objects are defined in terms of the events in which they participate. The temporal attributes of events have values from special temporal datatypes. We assume that these datatypes support the concept of *Time Intervals*, which are composed of *Time Points*. Time points could be represented as real numbers and Time Intervals as sets of real numbers. However, they could also be defined in other ways (we avoid making unnecessary ontological commitments at this point).

Finally, we would like to point out that a *process*, such as a chemical process or a business process, is a complex event.

V. UNIVERSALS

Universals *classify* individuals, which are said to be their *instances*. The set of all instances of a universal is called its *extension*. We consider seven kinds of universals: *event types*, *object types*, *quality universals*, *attributes*, *relator universals*, *reference properties* and *material relationship types*. There are other kinds of universals, but these seven are the most relevant for modeling and simulation

Universals *classify* individuals, which are said to be their *instances*. The set of all instances of a universal is called its *extension*. As depicted in Fig. 4, we consider seven kinds of universals: *quantity types*, *event types*, *object types*, *quality universals*, *attributes*, *relator universals*, and *material relationship types*. There are other kinds of universals, but these seven are the most relevant for conceptual modeling.

While the notions of attribute, relationship type and material reference property are well-known in computer science in the area of information and database modeling, their ontological foundation in connection with quality universals and relator universals is not well-known.

A universal U_1 is a *subtype* of another universal U_2 if the intension of U_1 (its set of features) includes the intension of U_2, and the extension of U_1 is included in the extension of U_2. Unlike data types and sets, universals do not have a standard set-theoretical *extensional semantics*. That is, if O_1 and O_2 denote two object types that happen to have the same extension, they still do not denote the same universal.

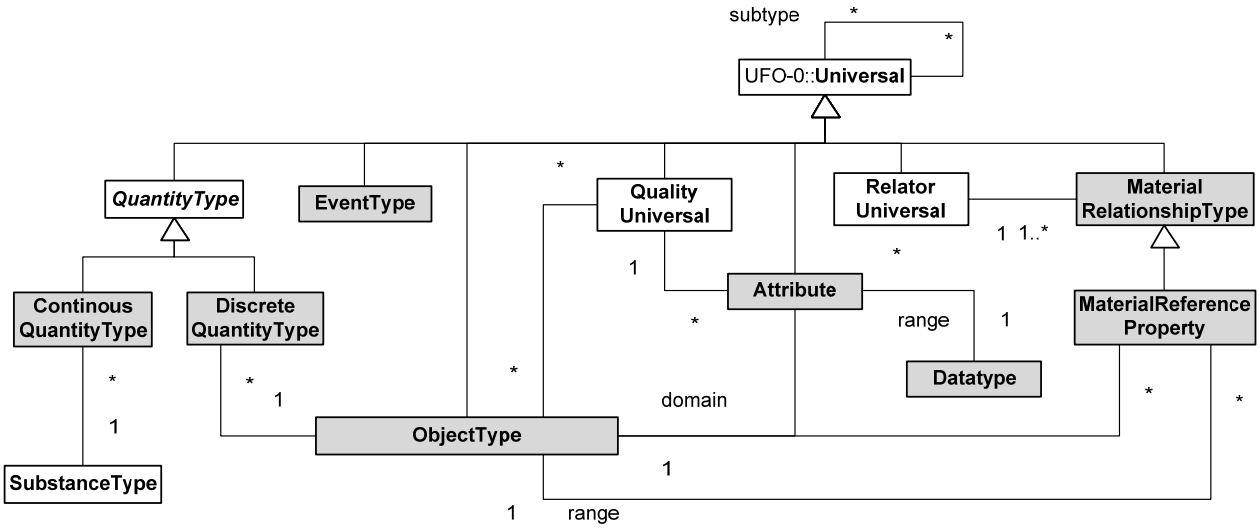

Figure 6. The ontology of universals eUFO-U (relevant terms in grey).

A. *Quality Universals and Attributes*

A ***quality universal*** classifies individual qualities of the same type. A quality universal can be associated with one or more data types, such that any particular quality corresponds to a specific data value from the value space of the data type. The association between qualities from some quality universal and the corresponding data values from an associated data type is provided by an ***attribute***, which is a universal that classifies *attributions*. A quality universal can be captured by one or more corresponding attributes, each of them based on a different data type.

E.g., the quality universal "hair color" could be captured by an attribute with the range of RGB byte triples or by an attribute with the range of natural language color names. Consequently, we may have more than one attribution for a particular quality, one for each associated attribute.

B. *Relator Universals, Material Relationship Types and Material Reference Properties*

A relator universal classifies individual relators of the same type. The ***material relationship type*** *R* induced by a relator universal ***R*** classifies all material relationships induced by relators from ***R***. Since each material relationship corresponds to a tuple, *R* also has a *tuple extension* (i.e. a *relation* in the sense of set theory). A material relationship type is a universal that classifies material relationships, which are 'truth makers' for material relationship statements.

A ***material reference property*** represents a binary material relationship type, corresponding to a relator universal whose instances mediate exactly two individuals. Its tuple extension is a subset of the Cartesian product of the extensions of the two involved types. The first type is called the ***domain***, and the second one the ***range*** of the reference property.

VI. Different Kinds of Object Types

We distinguish between the different kinds of object types shown in Fig. 7.

A. *Sortal Types and Mixin Types*

While all object types carry a principle of application, only *sortal types* carry a principle of identity for their instances. A principle of application allows to judge whether an individual is an instance of that object type. In contrast, a principle of identity allows to judge whether two individuals are the same.

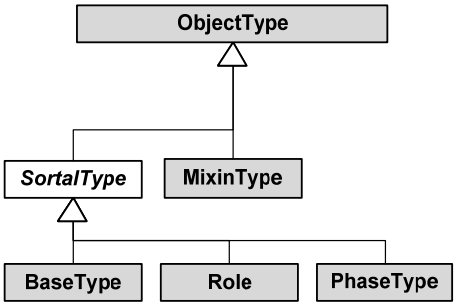

Figure 7. Substance individuals. (eUFO-A)

As an illustration of this point, contrast the two object types Apple and RedThing instantiated by two individuals *x* and *y*: both object types supply a principle according to which we can judge whether *x* and *y* are classified under those types. However, only the object type Apple supplies a principle which allows to decide whether *x* and *y* are the same (i.e., merely knowing that *x* and *y* are both red gives no clue to decide whether or not *x=y*). Non-sortal types, such as RedThing, are called *mixin types*.

B. Base Types

Within the category of sortal types, we make a further distinction based on the formal notions of rigidity and anti-rigidity: A sortal type U is *rigid* if for every instance x of U, x is necessarily (in the modal sense) an instance of U. In other words, if x instantiates U in a given world w, then x must instantiate U in every possible world w'. In contrast, a sortal type U is *anti-rigid* if for every instance x of U, x is possibly (in the modal sense) not an instance of U. In other words, if x instantiates U in a given world w, then there must be a possible world w' in which x does not instantiate U. We call a rigid sortal type a *base type*.

An example that highlights this distinction is the difference between the base type Person and the anti-rigid object types Student and Adolescent instantiated by the individual John in a given circumstance. While John can cease to be a Student and Adolescent, he cannot cease to be a Person. In other words, while the instantiation of the object types Student and Adolescent has no impact on the identity of an individual, if an individual ceases to instantiate the base type Person, then she ceases to exist as the same individual.

C. Roles and Phase Types

John can move in and out of the object type Student, while being the same individual, i.e. without losing his identity. This is because the principle of identity that applies to instances of Student and, in particular, that can be applied to John, is the one which is supplied by the base type Person of which Student is a subtype. For any anti-rigid object type A, there is a unique ultimate base type B, such that: (i) A is a subtype of B; (ii) B supplies the unique principle of identity obeyed by the instances of A. There is a specialization condition SC such that x is an instance of A iff x is an instance of B that satisfies SC. A further clarification on the different types of specialization conditions allows us to distinguish between two different kinds of anti-rigid object types: *role types* and *phase types*. Phase types constitute possible stages in the history of an individual. Examples include: (a) Alive and Deceased: as possible stages of a Person; (b) Catterpillar and Butterfly of a Lepidopteran; (c) Town and Metropolis of a City; (d) Boy, Male Teenager and Adult Male of a Male Person.

Role types differ from phase types with respect to the specialization condition SC. For a phase type P, SC represents a condition that involves only intrinsic properties of P. For instance, one might say that if John is a Living Person then he is a Person who has the property of being alive or, if Spot is a Puppy then it is a Dog that has the property of being less than one year old. For a role type R, conversely, SC involves extrinsic (relational) properties of R. For example, one might say that if John is a Student then John is a Person who is enrolled in some educational institution (playing the role of a student), if Peter is a Customer then Peter is a Person who buys a Product x from a Supplier y (playing the role of a customer), or if Mary is a Patient than she is a Person who is treated in a certain medical unit (playing the role of a patient).

For any role type (e.g. `Student`) there is an underlying binary *material relationship type* or *reference property* (e.g. `students`) such that the extension of the role type (e.g. the set of current students of an educational institution) is the *range* of that reference property.

In [5], it is formally proven that the following constraints hold:

1. Every object must instantiate exactly one ultimate base type.
2. A rigid object type cannot be a subtype of an anti-rigid object type (e.g., Person cannot specialize Student).
3. A mixin type cannot be specialized by a sortal type (e.g., Person cannot specialize Customer).
4. A mixin type cannot have direct instances.

D. Example

We illustrate these conceptual distinctions with the help of an example, shown in Fig. 8.

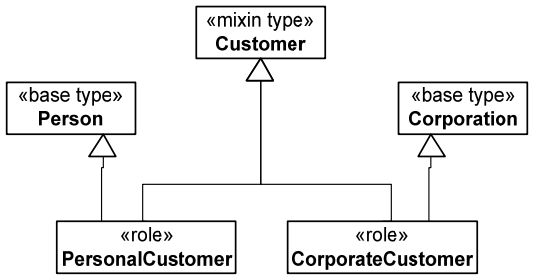

Figure 8. Example: Customer as a mixin type

Modeling the object type Customer is a non-trivial problem. Often, people are confused about the questions: is a customer a person? Or, conversely, is a person a customer? Neither of these subclass relatioships holds. Rather, Customer is a mixin type that can be segmented in two role subtypes: PersonalCustoper and CorporateCustomer, which specialize the base types Person and Corporation.

VII. Considering Agents as Special Objects

Agents are special objects. We want to consider all kinds of living beings, including insects and bacteria, as agents. We also want to consider certain artificial systems (such as robots) and social systems (such as organizations) as agents. On the other hand, we want to exclude all kinds of passive objects, such as chairs, apples and mountains, from our concept of an agent. So, what is common to living beings, robots and social systems? We claim that all these objects are *interactive systems* that are able to *interact* with passive objects in their environment or with each other in a *purposeful* way. The question what constitutes *interaction* is closely related to the question of what is an *action*.

We define *actions* to be those events that are the direct result of the *purposeful behavior* of an *interactive system*. Notice that this definition does not exclude higher-level action concepts such as *intentional actions*, which we just consider to be a special case of our general action concept. So, we do not require an agent to have a mental state with 'desires' and 'intentions', as it is common in many Artificial Intelligence approaches to multi-agent systems, in particular in the popular *Belief-Desire-Intention* (BDI) approach.

It is obvious that we have to include in our account of interactive systems the concepts of *perception* and *action*. We conceptualize both of them as special kinds of events: *perception events* and *action events*, as depicted in Fig. 9. For being able to model communication as a special kind of interaction between agents, we introduce the concepts of a *message* and a *communication event*, see Fig. 10.

The influence of perceptions on the actions of an agent is given by its reactive behavior, which is based on behavior patterns in the form of *reaction rules*. A perception event may lead, via a reaction rule, to a resulting action of the agent in response to the event, or to an update of the agent's information state, which may include *belief*s. We assume that beliefs are expressed as *belief statements* in the agent's belief representation language.

The influence of actions, and other events, on the perceptions of an agent is given by the causal laws of the agent's environment, taking the form of *transition rules*, (depicted in Fig. 11), which determine the caused perception events.

Beliefs are part of the agent's *information state*, which is the agent's basis for making action decisions. Simple belief statements have the form of entity-property-value triples, which are special cases of atomic sentences of predicate logic.

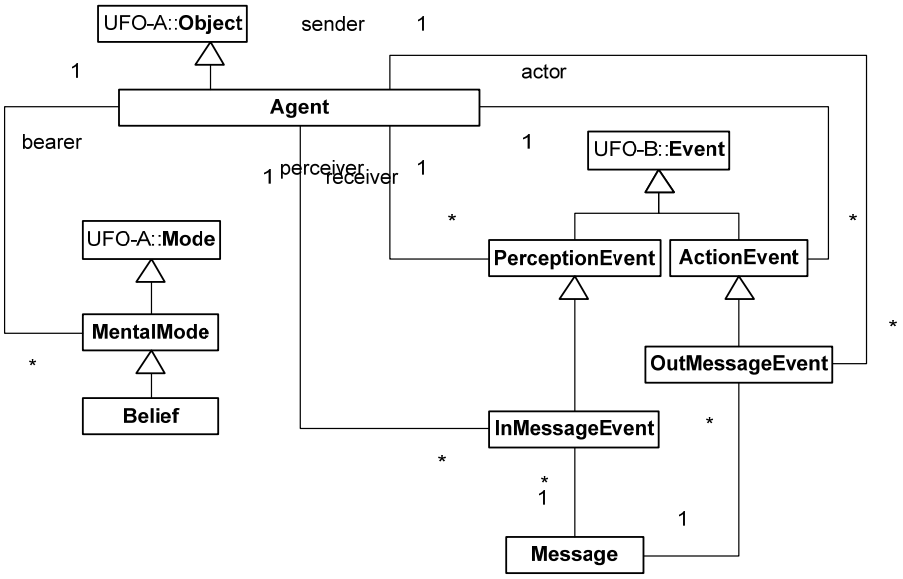

Figure 9. The ontology of agents (UFO-C)

A. Communication Events

A *communication event* is a complex event, having one sender and one or more receivers. It binds an *out-message event* to one or more *in-message events* (one for each receiver), sharing a common outgoing and incoming message, as described in Fig. 10.

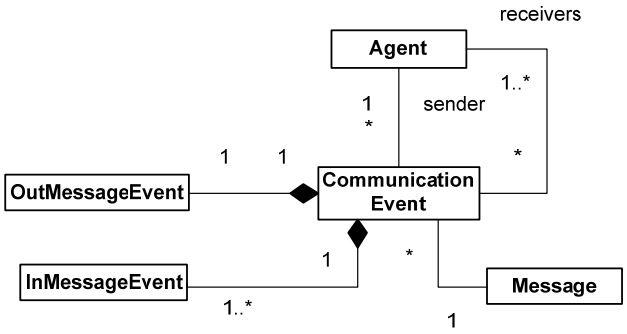

Figure 10. Communication events

VIII. The Categories of DESO and ABDESO

In modeling practice, we need more flexibility than a rigorous foundational ontology such as UFO would allow. For instance, we often want to "objectify" a relationship type by modeling it as an object type. Or we want to abstract away from the physics properties of an object type. Therefore, whenever we want to have explicit physics support, we define an entity type as a physical object type, otherwise as a plain object type.

This leads to the type concepts shown in Fig. 11, where we define the mixin category *entity type* as a super concept that subsumes both object type and event type. Fig. 11 defines the basic type concepts of the DES ontology *DESO* proposed in [6]. DESO, which is based on eUFO, is tailored to the domain of discrete event systems.

Notice that also the concept of a *transition rule*, corresponding to the well-known concept if a transition function, is included in the type categories of DESO. These rules have the form of an event-condition-action rule. They define the state change and event causation patterns that govern the transitions of a discrete event system.

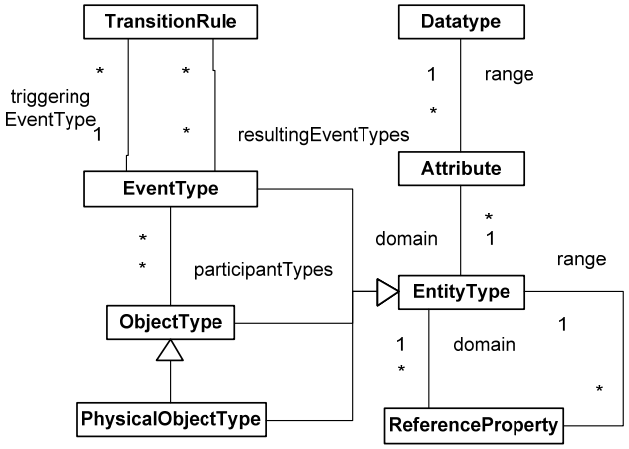

Figure 11. The type concepts of DESO

In addition to these type concepts, DESO comprises the individual concepts shown in Fig. 12.

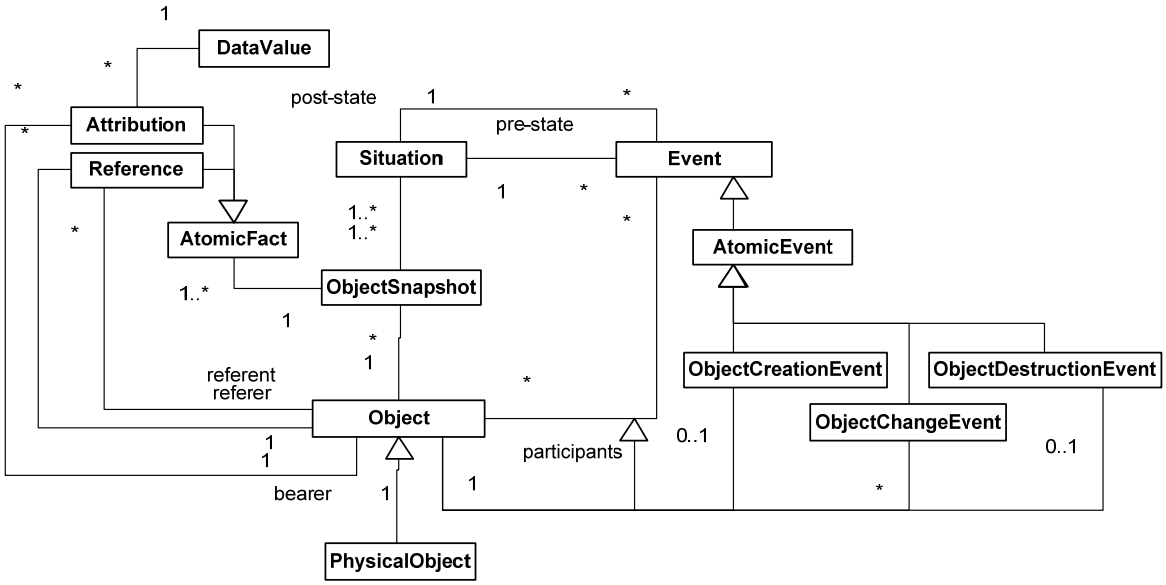

Figure 12. The individual concepts of DESO

ABDESO extends DESO by adding the agent-related concepts shown in Fig. 9 and 10, and the concept of ***institutional agents*** shown in Fig. 13. ABDESO allows specifying reactive behaviors per agent type in the form of ***reaction rules***, which can be viewed as special transition functions, triggered by perception events and resulting in state changes and follow-up action events. An institutional agent, such as an organization, may define roles to be played by its subagents. These roles, as agent types, define additional behaviors for its instances. The overall behavior

of a subagent of an organization results from merging all reaction rules associated with the agent's base type and all the roles played by it. Since roles can be assumed and dropped at runtime, this leads to a dynamic behavior definition for agents.

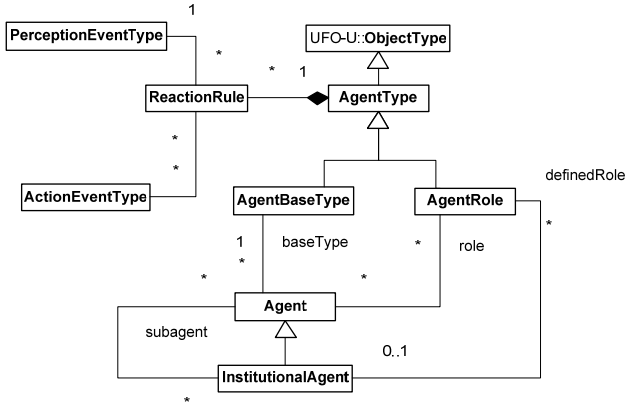

Figure 13. Institutional agents

IX. Evaluating Modeling and Simulation Languages against an Ontology

Both IS modeling languages and DES simulation languages can be evaluated by comparing a representation of their concepts, typically provided by a metamodel of the language, to an ontology of discrete event systems (more precisely, to an ontology that represents a shared conceptualization of the domain of discrete event systems). The stronger the match between them, the easier it is to communicate and reason with models made in that language.

A system model is made for representing a conceptual abstraction of the real world system under consideration. Such an abstraction of a system is formed on the basis of a conceptualization of the system's domain. Both for information system (IS) engineering and for system simulation engineering, we deal with the domain of *discrete dynamic systems*, also called *discrete event systems (DES)*. In order for a DES model *M* to faithfully represent a DES abstraction *A*, the DES modeling language *L* used to make *M* should be faithful with respect to the conceptualization of DES used to form *A*.

For a given ontology of discrete event systems O and a given IS or DES modeling language *L*, we may consider (1) a *representation* mapping from the concepts of O to the elements (or modeling primitives) of *L;* and (2) an *interpretation* mapping from the elements of *L* to the concepts of O. If these mappings are far from being isomorphisms, this indicates soundness and completeness problems of *L*. Notice that this approach does not depend on our proposals of DESO and ABDESO. It works in general for any kind of DES ontology O.

There are four properties of a simulation language to be checked in its evaluation:

1. ***Soundness***: *L* is sound wrt O iff every element of *L* has an interpretation in terms of a domain concept from O. The ***degree of soundness*** can be measured relatively as the number of *L* elements that have an O interpretation divided by the total number of *L* elements.
2. ***Completeness***: *L* is complete wrt iff every O concept is represented by a modeling primitive of *L*. The ***degree of completeness*** can be measured relatively as the number of O concepts that are represented by an element of *L* divided by the total number of O concepts.
3. ***Lucidity***: *L* is lucid wrt O iff every element of *L* has at most one interpretation in O. The ***degree of lucidity*** can be measured relatively as the number of *L* elements that have at most one interpretation in O divided by the total number of *L* elements.
4. ***Laconicity***: *L* is laconic wrt O iff every domain concept from O is represented by at most one element of L. The ***degree of laconicity*** can be measured relatively as the number of O concepts that are represented by at most one element of *L*.

The lower these degrees are for a given language L, the more problems may be expected from using a model expressed in L, e.g. by communicating incorrect information and inducing the user to make incorrect inferences about the semantics of the model.

For showing the applicability of the proposed evaluation method, we now summarize some preliminary results obtained from evaluating some DES modeling and simulation languages and the agent-based simulation language *Brahms*.

A. Evaluating Classical Petri Nets

The formalism of classical *Petri Nets (PN)* provides an abstract process modelling language with an intuitive and elegant graphical syntax. But it is too abstract for being used in practice. However, it has been extended in several ways to allow using it for business process modelling and other forms of DES modeling.

When we evaluate PN against DESO, we obtain the benchmarks listed in Table 1. They show that PNs have only one deficiency: they are too incomplete for being used as a system modelling language. This result is not surprising. The soundness of PN is the basis of its success, and its incompleteness is the reason why there are so many proposals how to extend it.

B. Evaluating Atomic DEVS

The DEVS family of formalisms [8] is the result of a research effort for establishing a general discrete event simulation framework. The evaluation of the language of Atomic DEVS against DESO (see Table 1) shows that Atomic DEVS. has an even greater completeness problem than Petri Nets. Both formalisms lack support for state structure modelling concepts.

C. Evaluating SIMAN/Arena

The widely used *Arena* simulation tool is based on the simulation programming language *SIMAN* that is tailored to modelling production processes. Its evaluation in

Table I shows that it is also quite incomplete. This is mainly due to the fact that SIMAN is not a general purpose DES language, but has been tailored towards a specific "process simulation" view.

TABLE I. EVALUATION AGAINST DESO

Criteria	PN	Atomic DEVS	ARENA
Soundness	100%	100%	100%
Completeness	43%	21%	52%
Lucidity	100%	100%	100%
Laconicity	100%	95%	95%

D. Evaluating Brahms

Brahms [9] is an agent-based modeling and simulation environment a) for developing simulations of people, organizations, and objects such as tools, documents and systems; and b) for designing, simulating and implementing multi-agent software systems. The preliminary results of evaluating Brahms against ABDESO in Table II show that Brahms is quite incomplete, mainly due to the fact that Brahms does not support explicit event modeling.

E. Evaluating BPMN

The preliminary evaluation result obtained in [10], and shown in Table II, suggests 100% soundness indicating that the core elements of BPMN have been well-chosen. However, the results of only 60% completeness, 70% lucidity, and 32% laconicity suggest that there are quite a few ***missing*** concepts, ***ambiguous*** elements, and ***redundant*** elements in BPMN.

F. Evaluating AORSL

The Agent-Object-Relationship Simulation Language (AORSL) extends the Entity-Relationship Simulation Language (ERSL) by adding agent concepts (see www.AOR-Simulation.org). ERSL includes a form of transition rules (called 'environment rules'), while AORSL adds ***reaction rules***, which are transition functions for the local/subjective state of an agent.

TABLE II. EVALUATION AGAINST ABDESO

Criteria	Brahms	BPMN	AORSL
Soundness	100%	100%	100%
Completeness	43%	60%	91%
Lucidity	90%	70%	100%
Laconicity	100%	32%	100%

The evaluation results in Table II show that, compared to Brahms and BPMN, only AORSL is nearly complete.

X. CONCLUSIONS

Using the *Agent-Based Discrete Event Simulation Ontology (ABDESO)*, extending the *Discrete Event Simulation Ontology (DESO)*, which is derived from the *Unified Foundational Ontology (UFO)*, we have evaluated a number of basic and agent-based discrete event system modeling and simulation languages, including the rule-based language AORSL. The result of this preliminary evaluation is that AORSL, as an enterprise modeling and simulation language, is much more complete compared to Petri Nets and BPMN.

ACKNOWLEDGMENT

The author wants to thank Giancarlo Guizzardi for the enduring fruitful collaboration over the last 10 years.

REFERENCES

[1] G. Guizzardi and G. Wagner, A Unified Foundational Ontology and some Applications of it in Business Modeling. In: Janis Grundspenkis and Marite Kirikova (Eds.), Proc. CAiSE'04 Workshops. Faculty of Computer Science and Information Technology, Riga Technical University, Riga, Latvia. June 7–11, 2004. Volume 3. pp. 129–143. Riga, Latvia.

[2] G. Guizzardi and G. Wagner, Towards Ontological Foundations for Agent Modeling Concepts using UFO. In Agent-Oriented Information Systems (AOIS), selected revised papers of the Sixth International Bi-Conference Workshop on Agent-Oriented Information Systems 2005. Lecture Notes in Computer Science, volume 3508, 2005, pp. 110–124, Springer Berlin/Heidelberg.

[3] G. Guizzardi and G. Wagner, Using the Unified Foundational Ontology (UFO) as a Foundation for General Conceptual Modeling Languages. In: Poli, R. (Ed.), Theory and Application of Ontologies. Springer Berlin/Heidelberg, 2010.

[4] G. Guizzardi, R.A. Falbo, and R.S.S. Guizzardi,. Grounding Software Domain Ontologies in the Unified Foundational Ontology (UFO): The Case of the ODE Software Process Ontology. In: XI Ibero-American Workshop on Requirements Engineering and Software Environments (IDEAS'2008), 2008, Recife.

[5] G. Guizzardi, Ontological Foundations for Structural Conceptual Models, PhD Thesis, University of Twente, The Netherlands, 2005.

[6] G. Guizzardi and G. Wagner., Towards an Ontological Foundation of Discrete Event Simulation. In: B. Johansson, S. Jain, J. Montoya-Torres, J. Hugan and E. Yücesan (eds.). Proceedings of the 2010 Winter Simulation Conference. December 2010, Baltimore, Maryland, USA, pp. 652–664. Available via <http://www.informs-sim.org/wsc10papers/059.pdf>.

[7] G. Guizzardi and G. Wagner., Towards an Ontological Foundation of Agent-Based Simulation. In: S. Jain, R.R. Creasey, J. Himmelspach, K.P. White, and M. Fu (eds.). Proceedings of the 2011 Winter Simulation Conference. December 2011, Phoenix, Arizona, USA.

[8] B. Zeigler, Theory of Modeling and Simulation, Wiley Interscience, New York, 1976.

[9] M. Sierhuis, Modeling and Simulating Work Practice. BRAHMS: a multiagent modeling and simu-ation language for work system analysis and design, Ph.D. thesis, Social Science and Informatics (SWI), University of Amsterdam, SIKS Dissertation Series No. 2001-10, Amsterdam, The Netherlands, ISBN 90-6464-849-2.

[10] G. Guizzardi and G. Wagner., Can BPMN Be Used as a Simulation Modeling Language?,. in J. Barjis, T. Eldabi, and A. Gupta (Eds.), Enterprise and Organizational Modeling and Simulation, Lecture Notes in Business Information Processing, Volume 88, Springer-Verlag Berlin Heidelberg, 2011.