ACSI – Artifact-Centric Service Interoperation

Deliverable 1.3
The complete ACSI artifact paradigm

Project Acronym	ACSI	
Project Title	Artifact-Centric Service Interoperation	
Project Number	257593	
Workpackage	WP1 – ACSI artifact paradigm and interoperation hub framework	
Lead Beneficiary	UNIROMA1	
Editor(s)	Diego Calvanese	FUB
	Giuseppe De Giacomo	UNIROMA1
	Domenico Lembo	UNIROMA1
	Riccardo De Masellis	UNIROMA1
	Paolo Felli	UNIROMA1
	Domenico Fabio Savo	UNIROMA1
	Marco Montali	FUB
	Fabio Patrizi	UNIROMA1
Contributor(s)	David Boaz	IBM Haifa
	Babak Bagheri Hariri	FUB
	Francesco Belardinelli	Imperial
	Pieter De Leenheer	Collibra
	Massimiliano de Leoni	TU/e
	Marlon Dumas	UT
	Richard Hull	IBM T.J. Watson
	Maurizio Lenzerini	UNIROMA1
	Riccardo Rosati	UNIROMA1
Reviewer(s)	Lior Limonad	
Dissemination Level	PU	
Contractual Delivery Date	15/01/2012	
Actual Delivery Date	15/01/2012	
Version	1.2	

The research leading to these results has received funding from the European Community's Seventh Framework Programme [FP7/2007-2013] under grant agreement no. 257593.

Abstract

In this document we complete the description of the ACSI Artifact Paradigm. Specifically the ACSI Artifact Paradigm can be represented with three layers:

- *The Artifact Layer*: in an instantiation of the paradigm, this consists of an artifact schema and a set of artifact instances that the system manages.
- *The Realization Layer*: in an instantiation of the paradigm, this consists of the actual services for storing and managing data, executing and recording actions, and interfacing with the external world, all according to the artifact schema.
- *The Semantic Layer*: in an instantiation of the paradigm, this consists of a conceptual specification of the static and dynamic aspects of the domain of interest.

Deliverable D1.1 concentrated on the Artifact Layer and on the Realization Layer. The *Semantic Layer* is the main subject of the current deliverable. In particular, we give here a detailed presentation of the Semantic Layer, we report on the concepts at the base of the Semantic Layer, and we put in context the technologies and results that are needed to support it.

Document History

Version	Date	Comments
V1.0	23-10-2011	First draft
V1.1	16-12-2011	Second draft
V1.2	15-01-2012	Final version

Table of Contents

List of Figures

1 Introduction

An *ACSI interoperation hub* serves as the anchor for a collaboration environment, that is, an IT environment used to support large numbers of service collaborations that operate independently, but which focus on essentially equivalent common goals. Unlike orchestrators, an interoperation hub works well in the context of open service networks. These hubs are primarily reactive, serving as a kind of structured white board that participating services can refer to, that can be updated with information relevant to the group, that can assist the services by carrying out selected tasks, and that can notify services of key events. Details on the ACSI interoperation hub can be found in Deliverable D1.2 & D1.4.

Artifacts are used to provide the underlying model of operations and processes in an interoperation hub. A (dynamic) artifact is a key conceptual entity that evolves as it moves through a business (or other) process. Artifacts provide a holistic marriage of data and process, both treated as first-class citizens, as the basic building block for modeling, specifying, and implementing services and business processes. An *artifact type* includes both a data schema and a lifecycle schema, which are tightly linked. The data schema provides a representation of the information that is relevant for the artifact. The lifecycle schema specifies the different ways that an artifact instance might evolve as it moves through the overall process.

In the context of single enterprises, industry, and in particular IBM, has shown in the last years that the use of artifacts can lead to substantial cost savings in the design and deployment of business operations and processes, and can dramatically improve communication between stakeholders, especially in cases where they represent different "silos" of the enterprise [17, 134, 142]. Artifacts can give an end-to-end view of how key conceptual business entities evolve as they move through the business operations, in many cases across two or more silos. As a result, artifacts can substantially simplify the management of "hand-off" of data and processing between services and organizations. A key pillar of the ACSI research is to scientifically investigate artifacts in the context of interoperation hubs and service collaborations. Much of the work on artifacts, up to now, has been guided by pragmatical consideration. In ACSI we aim at isolating the foundational elements of an artifact-based approach, and study them from a formal point of view. Obviously, in doing so, we will still be deeply concerned with pragmatical issues, both in the implementation and in pilot use cases where the strength of this approach can unfold. Also, much of the work on artifacts to date has used a form of finite state machines to specify the artifact lifecycle schemas. In ACSI, we pursue a more declarative approach to lifecycle specification that is more flexible and adaptive to variation and changes in the modeled processes. In the context of the interoperation hub, various tasks, traditionally seen as occurring within the states of state machines, are instead performed by the various participating services (although the interoperation hub might perform some of these tasks itself). Furthermore, it is typically the services that advance the lifecycle from one state to another when certain conditions, i.e., guards, on the artifact instances data are met. The artifact types used to specify the data and process management in an interoperation hub also provide a natural basis for specifying the access privileges of the different participating services.

As shown in Figure 1, the components of the scientific research stream of ACSI can be summarized into: ACSI Artifact Paradigm, ACSI Interoperation Hub Framework, Formal-based Techniques and Tools, and Observation-based Techniques and Tools. In this deliverable we concentrate on the first component of the ACSI Artifact Paradigm, which concerns the development of a formal, logic-based artifact paradigm that can support reasoning about, interacting with, and the evolution of artifact types, especially as they arise in interoperation hubs. Its development contributes to the achievement of some of the ACSI specific objectives, and in particular to (i) the development and exploitation of declarative approaches for specifying the lifecycles of dynamic artifacts, to improve their applicability for use in interoperation hubs, and (ii) the development of a formal foundation for dynamic artifacts, including the development of a semantic layer, which will provide a principled basis for specifying and implementing views, transformation, and evolution of artifact schemas in the context of interoperation hubs. As

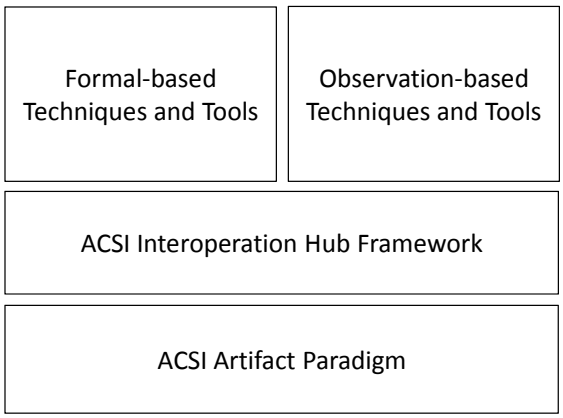


Figure 1 – ACSI Research stream

shown in Figure 2, the ACSI Artifact Paradigm can be represented with three layers:

- *The Artifact Layer*: in an instantiation of the paradigm, this consists of an artifact schema and a set of artifact instances that the system manages.
- *The Realization Layer*: in an instantiation of the paradigm, this consists of the actual services for storing and managing data, executing and recording actions, and interfacing with the external world, all according to the artifact schema.
- *The Semantic Layer*: in an instantiation of the paradigm, this consists of a conceptual specification of the static and dynamic aspects of the domain of interest.

Deliverable D1.1 concentrated on the Artifact Layer and on the Realization Layer only. The Artifact Layer is the core layer of the ACSI Artifact Paradigm. It provides the actual artifact types that are available in the system, each, in turn, characterized by the manipulated data and the processes that manipulate them. It is important to stress that the languages used to specify artifact types are high-level but executable. On the one hand, artifact types are concrete enough to make it possible to "compile" them into actual running programs, through suitable, model-driven techniques. On the other hand, artifacts abstract from most implementation details, providing a description of the activities of interest to the system at what we may call a "logical-level", borrowing the terminology from relational databases. We will use this level of abstraction in most of the ACSI research, including work on verification (WP2) and conformance testing (WP3).

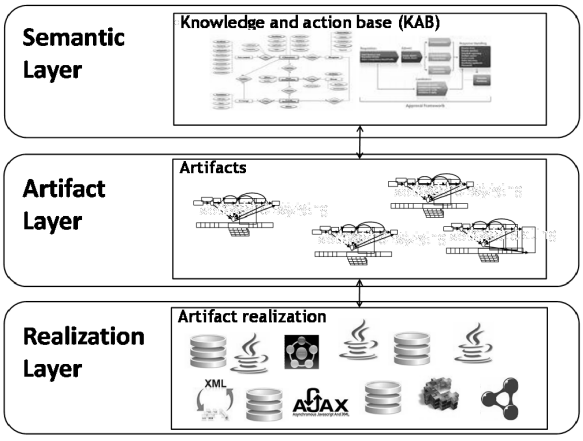


Figure 2 – ACSI Artifact Paradigm

The Semantic Layer remained out of the scope if D1.1 and it is the main subject of the current deliverable.

The specific contributions of this deliverable are the following:

1. *ACSI Knowledge and Action Base* (Section 3). We define the Semantic Layer in terms of what we call a *Knowledge and Action Base (KAB)*. The KAB main focus is to conceptually model the data that are present in the artifact system at the Artifact Layer, and their evolution as the system executes. Such a description is conceptual (or semantical) and is done in terms of the concepts and relationships that exists in the domain of interest of the interoperation hub, independently from the artifacts and other components that are operatively present at the Artifact Layer. The Artifact Layer is then connected to the Semantic Layer through formal mappings that relate data in the artifact systems and data in the KAB. Notice that this relation is rather sophisticated to reflect the different aims and granularity of the two layers. It is also important to understand that data are concretely present in the Artifact Layer but only virtually present in the Semantic Layer: the KAB does not store directly data, it contains the mappings that transform (on-the-fly) data at the Artifact Layer into data at the Semantic Layer. Such a conceptualization of data at the Semantic Layer and their relationship with the data ar the Artifact Layer forms the so called *knowledge component* of the KAB. Beside it we have the *action component* of the KAB, which deals with the dynamics of the domain of interest i.e. how it may possibly evolve as the result of actions. Notice that the focus is again on the domain of interest of the interoperation hub and not on the specific lifecycle, dynamic constraints, and processes of the artifacts and other components at the Artifact Layer, even if the two dynamics are indeed related. Specifically, given the mappings between the data in the two layers, executions at the Artifact Layer correspond to executions at the Semantic Layer. Such execution at the Semantic Layer must obey to the dynamics description of the domain of interest, specified in the action component of the KAB. If they fail to do so, then we know that there are problems with the corresponding artifact system execution at the Artifact Layer, because it violates the dynamics of the domain of interest. On the basis of this core observation we can develop a variety of services making use of the Semantic Layer.

   We also stress, however, that the artifact system may be defined and work without the Semantic Layer. The presence of the Semantic Layer gives the opportunity of bringing about and formalize relationships between the artifacts and the other component at the Artifact Layer that normally are in the mind of the designer. This ability, which is the core of the success of semantic technologies in governance of information systems, is particularly valuable in a system like the interoperation hub where multiple design from different organizations are deployed.

2. *Technologies for ACSI Knowledge and Action Base* (Section 4). We review the main technologies, research results, and research directions that can be adopted for realizing the KAB and Semantic Layer. Specifically we discuss:

   - *Ontologies*. These constitute the best technology available for conceptual representation. Current ontology languages are able to capture virtually all constructs typical of conceptual diagrams normally used in Software and Data design, such as UML Class Diagrams or Entity Relationship Diagrams. On the other hand they are formal enough to become an actual software system that supports a variety of automated verification and querying services.
   - *Description Logics*. These are the full-fledged logics underlying the best known ontology languages such as OWL DL or OWL2. They are typically well-behaved fragments of First-Order Logic and support decidable/effective reasoning tasks, which are at the base of all services typically associated with ontologies.

- *Ontology Based Data Access and Integration.* This is one of the hottest areas in ontologies: devise ontology languages/description logics that support query answering over large amount of data as one typically finds in relational databases nowadays.
- *Mappings.* As mentioned above, these are crucial for linking the KAB to the data at the Artifact Layer. Such kind of mappings have been studied and analyzed in detail in the literature on data integration and data exchange to map a so-called "global view of the data" to data sources. More recently, the use of ontologies for representing the global view as been strongly advocated by the scientific community.
- *First-order variant of temporal logics.* To realize the KAB action component we need to express dynamics of the domain of interest in terms of full-fledged temporal logics such as LTL, CTL, μ-calculus. However such logics are normally considered for verification in their propositional variant. Here we need to consider First-Order variants so as to deal with data, values and concept instances that will be present in the knowledge component of the KAB as resulting from the mappings application over the actual data in the artifact system. Dealing with such situation is very challenging in general and is one of the most important contributions that WP2 in ACSI is providing to the scientific community.
- *Semantic Log.* An interesting outcome of having a Semantic Layer is the possibility of using it for governing the interoperation hub. One important point is to query the current and past states of the interoperation hub, possibly to compare situations as well as for analysis and reporting purposes. This gives rise to the notion of Semantic Log, which can store (pieces of) the snapshots characterizing the historical evolution of the system at the Semantic Layer. We discuss various option on how one can log the system conceptually at the Semantic Layer.

3. *An example of Semantic Layer for the Order-to-Cash Scenario.* This is used to make the ideas reported in the document concretely visible on a familiar scenario.

The document is complemented by an initial review of the A^3M for the Artifact Layer, upon which we build the Semantic Layer. This is taken from D1.1 except for a final discussion on events and messages.

To have the complete picture of the ACSI Artifact Paradigm the current document must be considered together with D1.1. In other words D1.1 + D1.4 tighter detail the ACSI Artifact Paradigm used in the rest of the project. We decided to avoid including all the content of D1.1 here, being the document with new material already long.

In this document we complete the description of the ACSI artifact paradigm that we started studying in Deliverable D1.1, where the core ACSI artifact paradigm has been presented. In particular, we give here a detailed presentation of the *Semantic Layer*, which is posed on the top of the Artifact and Realization Layers, described in Deliverable D1.1.

2 Review of the ACSI Artifact Abstract Model

This section reviews the *ACSI Artifact Abstract Model*, also referred as A^3M, or *A-Cube Model*, which provides the conceptual and formal basis for the artifact layer of the ACSI Artifact Paradigm. The model is abstract since it is agnostic to the specific data model used for represent data and the specific process model used to represent processes. The ACSI Artifact Abstract Model introduces structure in Artifact Layer, which can be later exploited for designing, executing and verifying artifact-based systems. The content of this section is already present in Deliverable D1.1, we report it here since we refer to it in introducing the detail of the Semantic Layer. A new discussion subsection is however added at the end of the section, to make some consideration of the distinction between events and messages which is often made in previous work on business process models.

2.1 Artifact-Based System

An A^3M artifact-based system is formed by a set of interacting artifacts immersed in an environment, i.e., the external world, including users, external data sources, and services, in which the system runs. The environment manifests itself through specific sophisticated interfaces called environment gateways, through which data can be exchanged as well as stored and manipulated by both the artifacts of the system and external means. Artifacts and environment gateways are richly modeled using data schemas as well as static and dynamic constraints. A^3M is agnostic to the specific kind of data schemas adopted, however it allows for rich data representations, which we refer to as the databases of the artifacts. We assume that the mean to progress for the artifacts, gateways and more generally for the system is through "events". Such events are generated by the gateways, i.e., generated by the users of the system, as well as by artifacts as they react to received events. Notice that our notion of event is abstract and comprises aspects typically associated with messages in the current literature, such as a rich payload. Also we model explicitly how artifacts in the system are related to each other and to the environment gateways through a specific set of relationships. Notice that static and dynamic constrains can refer to such relationships if needed. A characterizing feature of artifacts is the so called "artifact lifecycle", which is constituted by dynamic constraints that describe how an artifact can evolve over time.

Formally, an A^3M *artifact-based system* is a tuple $\mathcal{AS} = (\mathcal{A}, \mathcal{V}, \mathcal{E}, \mathcal{R}, \mathcal{C}, \mathcal{D})$ where:

- $\mathcal{A}$ is a set of *artifact types*, called *artifact base*;
- $\mathcal{V}$ is a set of *environment gateway type*, sometimes called simply *environment*;
- $\mathcal{E}$ is a set of *event types*, called an *event base*;
- $\mathcal{R}$ is a set of *relationship types* involving artifact types, event types, and environment gateway types.
- $\mathcal{C}$ is a set of *static constraints* on the possible instantiations of $\mathcal{A}$, $\mathcal{V}$, $\mathcal{E}$, and $\mathcal{R}$, which must remain true at any point in time.
- $\mathcal{D}$ is a set of *dynamic constraints*.

We discuss each component separately below.

Artifacts.

An *artifact type* $A \in \mathcal{A}$ denotes a set of artifact instances with a common structure, specified by means of a data (or database) schema D_A. An artifact instance of type A is characterized by an artifact identifier (AID) and by a "database" conforming to the schema D_A. Artifact instances can be created and destroyed over time, and their database may evolve (remaining conformant

to the database schema of their artifact type), while the AID is immutable. Note that artifacts do not have predetermined associated operations, differently from classes in object-orientation. Indeed, the artifact database can be freely queried and updated, though dynamic constraints indirectly regulate the allowed changes.

Environment gateways.

Analogously to an artifact type, an *environment gateway type* $V \in \mathcal{V}$ denotes a set of gateways instances whose common structure is specified by means of a database schema D_V. A gateway instance of type V is characterized by a gateway identifier (VID) and by a "database" conforming to the schema D_V. Notice that the database of an environment gateway may evolve over time. The set of gateways forming the environment is used to remember relevant parts of the inputs and outputs that have been exchanged between the system and the outside world. In particular, this allows for modeling and supporting one-way and two-way service calls to the external world.

While formally gateways are very similar to artifacts, we will pose very different restrictions on them. Artifacts are essentially a reactive component that react in a predetermined way to events that it receives. Environment gateways are manifestation of the external world around the artifact systems. They are used to model the fact that users, humans or other systems take informed decision on how to proceed. So their behavior is highly nondeterministic when seen from the point of view of the artifact-based system. These differences show up in how we model artifact and environment gateways. For example we do not associate a "lifecycle" to gateways but only to artifacts (see later).

Events.

An *event type* $E \in \mathcal{E}$ denotes a set of event instances with a common structure, specified by means of a database schema D_E. An event instance of type E is characterized by an event identifier (EID) and by a database (i.e., payload) conforming to the schema D_E. Event instances can be created (generated) and destroyed (consumed) over time, however they are immutable, i.e., their payload and EID do not change once the event instance has been created. Again note that our notion of event is quite general and abstract and comprises aspects typically associated with messages in the current literature, such as a rich payload (it can be an entire database in principle).

Relationships and their stereotypes.

A *relationship type* $R \in \mathcal{R}$ among artifacts, environment gateways, and events, denotes a relationship between instances of the relationship components. The *relationship instances* (i.e., the tuples) may have attributes, for example, a *timestamp*.

We are interested in several key *stereotypes* (i.e., kinds) of relationships. In the following, we mention some stereotypes involving artifacts/environment gateways and events:

- *create-event* between an artifact type/environment gateway and an event type: a tuple (a, e) (possibly with timestamp t) in such a relationship denotes that the artifact instance a has created the (pending) event instance e (at time t).

  We require that for each e there exists exactly one a in a relationship of stereotype create-event.

- *may-read-event* (and *may-destroy-event*): a tuple (a, e) in such a relationship denotes that a may read (resp., destroy, i.e., consume) e.

- *read-event* (and *destroyed-event*): a tuple (a, e) (with timestamp t) in such a relationship denotes that a has (just) read (resp., destroyed) e (at time t).

  Observe that an event that is placed in a relationship of stereotype *destroyed-event* is waiting to be removed from the system and its payload cannot be accessed anymore.

- *selected-receiver*: a tuple (e, a) in such a relationship denotes that e has a as selected receiver.

  Observe: an event can have many selected receivers of a certain type, and can be involved in several relationships of this stereotype, thus allowing for receivers of different types.

  Observe: events that are broadcast are not involved in such relationships.

Also, we consider the following relationship stereotypes among artifacts/environment gateways:

- *create-artifact* between two artifact types: a tuple (a_1, a_2) (possibly with timestamp t) in such a relationship denotes that the artifact instance a_1 *has created* the artifact instance a_2 (at time t).

  We require that for each a_1 there exists at most one a_2 in a relationship of stereotype create-artifact.

- *may-destroy-artifact*: a tuple (a_1, a_2) in such a relationship denotes that the artifact instance a_1 may destroy the artifact instance a_2.

- *references*: a tuple (a_1, a_2) in such a relationship denotes that the artifact instance a_1 references the artifact instance a_2. This indicates that a process accessing a_1 can then follow the reference and access a_2.

  Different kinds of access are also of interest, for example read-only, read-write, read-only certain parts of the referenced artifact-database, etc.

- *referential-integrity/existence dependency*: a tuple (a_1, a_2) in such a relationship denotes that, if a_1 is deleted, then a_2 should be automatically deleted as well. This will be enforced through an (inter-artifact) dynamic constraint (see below).

We can actually write a catalog of the relationship types of interest, depending on the specific model that is used to instantiate the ACSI Artifact Paradigm. Such a *catalog* would contain for each stereotype the following relevant information:

- the types of involved artifacts/environment gateways and events;
- static constraints on the tuples in the relationship;
- intra- and inter-artifact dynamic constraints induced by the stereotype, see e.g., referential integrity above.

For example, in the GSM concrete model (see Deliverable D1.1.) one could use these stereotypes to constrain how guards and milestones can be specified.

Notice that, in general, there will be several relationship types for each of the stereotypes. For example, to distinguish the different ways in which an artifact may access another artifact, e.g., to read certain parts of a database or to write other parts.

Static constraints.

Static constraints on the system are constraints that must hold at every point in time during the execution of the system. Possible forms of static constraints are: mandatory participation to relationships, functionality of relationships, or, more generally, multiplicity constraints on the participation to relationships, key constraints and functional dependencies on relationships disjointness and inclusion constraints, etc. They are explicitly asserted in $\mathcal{C}$.

Such constraints are expressed in a formal language that enables *automated reasoning*, in order to *verify their consistency* and that *processes* acting on the artifact system don't violate them. We envision that most such constraints will not deal with the data in the database of the various artifact/gateway/event instances, though some may. Notice that in any case the database schema in those may in turn include local constraints on the data. On the other hand, if a Semantic Layer is introduced, this will specify sophisticated forms of constraints on the data in databases of the artifact/gateway/event instances.

2.2 Snapshots and Traces

We are interested in how the instantiation of the artifact-base system evolve over time, that is how artifact instances are created, modified and destroyed, how events are exchanged how environment gateways are updated. To capture this formally we introduce the key notions of "snapshot" and of "traces".

Snapshots.

A *snapshot* of the system $\mathcal{AS}$ is a function $\mathcal{I}$ that associates:

- to each *artifact type* A, a set $A^{\mathcal{I}}$ of artifact instances, and to each *artifact instance* $\alpha \in A^{\mathcal{I}}$ a database instance $\alpha^{\mathcal{I}}$ for schema D_A. We call $\alpha^{\mathcal{I}}$ an *artifact instance snapshot* of A;
- to each *event type* E, a set $E^{\mathcal{I}}$ of event instances, i.e., the *pending* events, and to each such *event instance* $\varepsilon \in E^{\mathcal{I}}$ a database instance $\varepsilon^{\mathcal{I}}$ for schema D_E. Observe that, if for two different snapshots $\cdot^{\mathcal{I}_1}$ and $\cdot^{\mathcal{I}_2}$ we have that $\varepsilon \in E^{\mathcal{I}_1}$ and $\varepsilon \in E^{\mathcal{I}_2}$, then $\varepsilon^{\mathcal{I}_1} = \varepsilon^{\mathcal{I}_2}$ (indeed, data in event instances are *immutable*);
- to each *environment gateway* V, a database $V^{\mathcal{I}}$ for schema D_V;
- to each *relationship type* R among n artifact types/environment gateway type/event types $A_1, \ldots, A_n$, an n-ary relation $R^{\mathcal{I}}$ contained in $A_1^{\mathcal{I}} \times \cdots \times A_n^{\mathcal{I}}$.

The snapshot must *satisfy the static constraints* in $\mathcal{C}$. For example, if the constraints are specified as logical formulas, such formulas must be *true* in the snapshot.

Traces.

A *trace* τ of an artifact system $\mathcal{AS}$ starting from a snapshot $\mathcal{I}_0$ of $\mathcal{AS}$ is a possibly infinite sequence $\mathcal{I}_0, \mathcal{I}_1, \ldots$ of snapshots of $\mathcal{AS}$. A trace satisfies the following *structural dynamic constraint* (*continuous existence of artifact and event instances*): For each artifact instance α of each artifact type A, if $\alpha \in A^{\mathcal{I}_i}$ but for some $j > i$, $\alpha \notin A^{\mathcal{I}_j}$, then for all $k \geq j$ we have $\alpha \notin A^{\mathcal{I}_k}$. Similarly for event instances.

Dynamic constraints.

Dynamic constraints are constraints on traces. Examples of such constraints are the dynamic constraint asserting that the static constraints must hold in every snapshot, as well as the structural dynamic constraints mentioned above.

Typically, we are interested in placing further constraints on traces. For example, if in a snapshot $\mathcal{I}_i$, an artifact instance α has created another artifact instance β (recorded as a tuple $(\alpha, \beta) \in R^{\mathcal{I}_i}$, for some relationship R of stereotype *create-artifact*), then $(\alpha, \beta) \in R^{\mathcal{I}_{i+1}}$, assuming that α and β still exist in $\mathcal{I}_{i+1}$.

Generally, we distinguish between:

- *inter-artifact dynamic constraints*, which may involve several artifact instances, possibly of different types, as well as events, environment gateways, and relationships;
- *intra-artifact dynamic constraints*, which involve a single artifact instance, and hence characterize the artifact from a dynamic point of view.

All the dynamic constraints mentioned above are examples of inter-artifact dynamic constraints. The most important intra-artifact dynamic constraints are the so-called *artifact lifecycles* (see below).

Notably, we have not included yet mechanisms to actually progress an artifact system from one snapshot to the next, i.e., we have not specified yet tasks and service executions. Several mechanisms are possible, like flows in the FSM concrete model, or guard-stage-milestones in GSM concrete model (see Deliverable D1.1).

2.3 Artifact Lifecycle

In the artifact-centric methodology, the discovery of the key business artifacts goes hand-in-hand with the discovery of the macro-level (artifact) *lifecycles*. In most cases the business stakeholders can describe this macro-level lifecycle in terms of key, business-relevant *phases* in the possible evolution of the artifact, from inception to final disposition and archiving.

Phases.

Formally, given an artifact type A of a system $\mathcal{AS}$, an artifact lifecycle for A is based on the notion of *phase*. Phases of A partition the set of possible databases instances for the schema D_A into a (possibly infinite) set of equivalence classes. Each *phase* is one such equivalence class. Notice that, given a snapshot $\mathcal{I}$ of the system M, we can identify for each artifact instance $\alpha \in A^{\mathcal{I}}$, the phase of A to which α belongs, namely the phase S such that $\alpha^{\mathcal{I}} \in S$. The fact that the phases are defined only in terms of the artifact instance database, and not in terms of the relationships, conforms with the foundational idea that all the relevant information regarding the evolution of artifacts should be contained in its database. This means that, if relevant information is obtained by navigating relationships, then such information is replicated locally in the artifact.

Often, phases are defined in a rather direct way, by introducing specific attributes in the artifact database that record explicitly the phase. (The partition is simply done according to the values stored in such attributes.) For example, one can have a single attribute recording the phase, as in the FSM concrete model, or a set of attributes to record the "milestones" already achieved, and the milestones yet to achieve, i.e., the open stages, as in *GSM*.

Lifecyle.

The *lifecycle* of an artifact type A is a specification of a set of constraints on the allowed sequencing of the phases traversed by its instances. Such constraints can be specified in different ways:

- in terms of an *abstract process* specified in terms of the phases, e.g., as a transition system in which the causes (i.e., tasks, actions) of the transitions are not recorded, or as a set of rules defined in terms of the phases of A;
- in terms of *logical/declarative formalisms*, e.g., temporal logic or dynamic logic, specified in terms of the phases of A.

Notice that, *logical/declarative formalisms* allow for specifications that, while they can be fulfilled by many processes, *do not correspond exactly to any executable process*, in general! For example, "eventually always happy" is a specification that can be implemented in many ways by suitable processes; however, there is no process that is as general as the specification itself.

Alternative: phase predicates.

Instead of specifying phases, one could equivalently resort to a set of *phase predicates* defined over the set of database instances for the schema A_D. The phase predicates induce a partition of the sets of database instances, where an element of this partition is constituted by all database instances that agree on all phase predicates. Such a partition can also be considered as a set of phases, however these are specified indirectly, in a way that can be exponentially more succinct than explicitly given phases. For example, milestones and open stages in GSM can be considered as phase-predicates.

When using phase predicates, the lifecyle of an artifact is a set of constraints expressed in terms of the phase predicates.

Observations.

We stress that lifecycles and dynamic constraints are *not* meant to be *executable*! They are *declarative specifications of how the dynamics* of the system should be handled by any process/processes we may set to execute on the system.

Notably, we have *not included yet mechanisms to actually progress* an artifact system from one snapshot to the next, i.e., we have not specified yet tasks and service executions. As mentioned, and illustrated in the Deliverable D1.1, several mechanisms are possible, like flows in the FSM concrete model, or guard-stage-milestones rules in the GSM concrete model.

An artifact-centric system progresses by means of *processes* running over it. Possible process specifications are:

- FSM's state-machines and flows (procedural),
- GSM's stages (rule based),
- Proclets-like process specification (Petri nets).

The ACSI Artifact Paradigm does not commit to a specific formalism for processes. In other words, it is agnostic with respect to the process specification. However, we do insist on a formalization of the transition relations to captures how the system runs from an abstract point of view.

2.4 System Run

While we do not give any specific formalism for processes, we assume that the system evolves through time as the result of a *nondeterministic transition relation* that changes the snapshot.

Transition relation.

The transition relation

$$\mathcal{F} \subseteq \mathit{Snaps}(\mathcal{AS}) \times \mathit{Snaps}(\mathcal{AS})$$

is such that, for each $(\mathcal{I}, \mathcal{I}') \in \mathcal{F}$, we have that $\mathcal{I}'$ is obtained from $\mathcal{I}$ by:

- consuming some events,
- creating new events,
- introducing new artifact instances,
- destroying existing artifact instances,
- changing the existing artifact instances databases, and
- changing the environment gateways instances.

In going from $\mathcal{I}$ to $\mathcal{I}'$, all static and dynamic constraints must be respected, and this has an impact on the structure of $\mathcal{F}$.

Runs.

A *run* ρ of system $\mathcal{AS}$ under the transition relation $\mathcal{F}$ is a trace $\rho = \mathcal{I}_1, \mathcal{I}_2, \ldots$ of $\mathcal{AS}$ such that $(\mathcal{I}_{i-1}, \mathcal{I}_i) \in \mathcal{F}$, for each $i \geq 1$. Notice that we are interested in transition relations $\mathcal{F}$ whose runs comply with the static and dynamic constraints of the artifact system. Notice that given to concsecutive snapshots $\mathcal{I}_{i-1}$ and $\mathcal{I}_i$ of a run ρ, we can easily single out the changes happened to a single artifact instance or to a gateway, by analyzing the differences between $\mathcal{I}_{i-1}$ and $\mathcal{I}_i$.

Observations.

The relation $\mathcal{F}$ is a very abstract mechanism to progress an artifact system from one snapshot to the next. In practice, we need to make such mechanism concrete, introducing tasks, services, etc. Such concrete mechanisms may actually differ in different settings, for example they differ in FSM and in GSM.

The transition relation $\mathcal{F}$ is typically obtained by composing several transition relations that deal with the various components of an artifact system, as induced by the running processes. One fundamental distinction is between the "moves" of the artifacts and those of the environment. Such a distinction leads to a subdivide of the transition relation into an *artifact transition relation* $\mathcal{F}_{\mathcal{A}}$ and an *environment transition relation* $\mathcal{F}_{\mathcal{E}}$, which respectively change the artifact, leaving the environment untouched, and vice-versa.

2.5 Discussion on A^3M Notion of Events

The description of event types and relationships, like for the other elements of the A^3M model, is intentionally kept as minimal as possible. In fact, the aim of A^3M is to elicitate a core set of key concepts needed to characterize artifact-based systems in general. Specialized artifact systems can then be built on top of this abstract model, using dedicated stereotypes and event types to introduce and properly describe the additional concepts needed to tackle the application domain at hand. For example, messages and timing events, typically used in a BPM scenario to drive the information exchange and define time-dependent behaviors, can be seamlessly modeled by refining the structure of event types and/or adding new relationship stereotypes.

This approach is in line with other recent conceptual modeling efforts in the BPM area. A notable example is *XES*, an XML-based format for the representation of event logs that has been recently adopted by the *IEEE Task Force on Process Mining* as reference standard (see `http://www.xes-standard.org/`). In order to "capture event logs from any background, no matter what the application domain or IT support", the XES meta-model consists of two parts:

- A small set of core concepts and constraints used to characterize the fundamental structure of an event log in an abstract way, i.e., that a log contains a set of traces, each of which in turn contains a set of events; at this level, no further information is attached to any of these concepts.
- A set of concepts that can be specialized on a per domain basis in order to refine and extend the event log meta-model, by defining (typed) attributes and classifiers on any level of the log hierarchy (log, trace and event); for example, the *lifecycle* extension introduces a transactional model for activities, linking events to transitions in such a lifecycle model, whereas the *organizational* extension deals with application domains in which events can be caused by human actors, and thus associates to each event the resource that has triggered it, together with the role and group of the resource.

Similarly to attributes, classifiers and extensions in XES, A^3M supports the possibility of refining the abstract framework through specific forms of event types and relationship stereotypes. More generally, A^3M shares the same guiding principles that have been followed in the development of XES: simplicity, expressivity, flexibility and extensibility.

3 ACSI Semantic Layer: Concepts

In this section we describe in detail the Semantic Layer and how it fits in A^3M. In a nutshell, the Semantic Layer is realized through the formal notion of *Knowledge and Action Base* (KAB), that will hold information about both the static and dynamic aspects of artifact types and instances.

The KAB contains a conceptual representation of all the data maintained in an artifact-based system, specified in terms of an *ontology*. Such an ontology is expressed using an ontology language that is suitable for data access and integration, such as those proposed in [41, 39, 7]. The ontology comprises the conceptual representation of the data stored in the artifact databases, the environment gateways, and the event instances. Beyond that, however, it establishes at the conceptual level different kinds of relationships between the data of different artifacts and environment gateways, possibly involving data in the payload of the exchanged events. The connection between the Artifact Layer and the Semantic Layer is provided through suitable mappings [108, 88, 124], that express by means of mapping queries how the data in the artifact and the environment gateways that constitute the Artifact Layer are related to (the virtual) data at the Semantic Layer. Such mappings are then exploited in different ways: on the one hand, to satisfy information requests that are posed in terms of the Semantic Layer, and that require accessing data at the Artifact Layer; on the other hand, to understand tasks and processes that specify how to modify data stored in artifact instances in terms of operations specified over the Semantic Layer [60, 50].

Also, one can exploit the Semantic Layer to model static and dynamic constraints that are important in the domain of interest of the system. In particular one can consider sophisticated properties expressed in, for example, temporal logics used for specification and verification of systems such as LTL, CTL, μ-calculus, etc. [71, 139, 70, 90, 111, 69]. Such properties expressed at the semantic level have images in A^3M as static and dynamic constraints, possibly involving data present in various artifacts and environment gateways. Such images can be used to verify various forms of conformance of new artifacts and processes introduced in the systems. Also more related to the specific use of the Artifact Framework in the context of the interoperation hub, one can formally assess to which piece of information the various stakeholders participating to the interoperation hub share an access. Finally, information about the semantic actions that are possible in the domain are representable at the Semantic Layer, including pre- and post-conditions concerning their impact, expressed in terms of the knowledge base; this is in the spirit of AI research on reasoning about actions [125, 129, 126]. These pre- and post-conditions can be used, for example, to assess which tasks are available, necessary, or missing for expressing processes of interest.

In what follow, we pass through each one of the notions introduced above, and provide a detailed and formalized description of them.

3.1 Knowledge and Action Base

The *Knowledge and Action Base* (KAB) provides a conceptual treatment of artifact-centric systems. In particular, its main role it to give a *semantical account of the data* characterizing the domain of interest as maintained in the system, specified in terms of a Description Logic *knowledge base* (or *ontology*). The ontology comprises an integrated conceptual representation of the data stored in the artifact databases, the environment gateways, and the events. This is of utmost importance, because while the A^3M provides details on artifacts, gateways, events, and how these are related to each other, it *does not specifically address the data* contained in them and what information such data carry along.

Alongside the conceptual representation of the artifact system data, the KAB also includes an *action specification* that implicitly characterizes the legal evolutions of the domain of interest as captured in such a knowledge base, and hence defines the behavioral boundaries that must be respected by processes defined in the Artifact Layer.

Understanding the semantics of data contained in the artifacts, gateways and events is important for a number of reasons, such as for example:

- To provide business managers and analysts with a *unified and well-structured view* of the whole information present in the system, enabling *governance of the data* at a high level of abstraction, and providing support for *querying* such information, which is the basis for *reporting* and *analysis*.
- To *relate different artifacts* that share information, though possibly with very different representation, in their artifact instances. (This tasks becomes even more important and challenging when artifacts belong to different organizations).
- To understand how a *new participant could take advantage of the information contained in available artifacts*, gateways and events of the system.
- To *discipline the introduction of new artifacts and processes* in the system, checking whether they seamlessly integrate with the already existing artifacts and processes, and supporting various forms of *conformance* tests.
- To handle *authorization views*, used in the context of the interoperation hub to formally regulate to which pieces of information the various stakeholders participating to the hub share an access.

Following recent literature on data integration, data are only virtually present in the KAB. During the execution, data are stored inside the artifacts and manipulated by processes defined in the Artifact Layer. Such data can be then homogeneously accessed through the KAB, which exports a conceptual, virtual view of the current snapshot of the system (and possibly of previous snapshots as well, see Section 4.8).

Formally, given an ontology language $\mathcal{L}$, a KAB is a tuple $\mathcal{KA} = (\mathcal{K}, \Gamma)$, where:

- $\mathcal{K}$ is the *knowledge component*, i.e., an ontology that contains complex descriptions of the concepts and their relationships of the domain of interest, expressed in $\mathcal{L}$. Beside such an ontology the knowledge component include mappings to extract information from the data in the databases of the artifact layer.
- Γ is the *action component*, composed of dynamic constraints that define the allowed evolutions of the KAB states.

We discuss each such component in detail next.

3.2 Knowledge Component and its Linkage with the Artifact Layer

The knowledge component represents the information on the domain of interest of an artifact-centric system. In particular, the knowledge component composed of two subcomponent $\mathcal{K} = (\mathcal{T}, \mathcal{M})$, described in the following.

TBox.

$\mathcal{T}$ is a so called TBox expressed in an ontology language based on Description Logic suitable for *Ontology Based Data Access* (OBDA) (see later). Such a TBox describes intensionally, at the conceptual level, the key concepts and relationships (roles) of the domain of interest. In other words, the knowledge component provides a *semantical characterization of databases of artifacts*, gateways, event payloads in terms of views over the semantic layer.

Notice that the use of a logic for capturing our knowledge on the domain of interest, typical of the work on Knowledge Representation in Artificial Intelligence, allows us to formally verify that interesting conditions or constrains on the relationships among data are enforced. It also

allows us to work with incomplete information, i.e., without assuming that the knowledge in the current artifacts is all we will ever know of the domain. This gives a flexibility and an elaboration tolerance that is often considered crucial in data integration tasks. Observe that this is a data integration task since data that are scattered among the various artifact are seen in a unified view at semantic level, being indeed the information about the same domain of interest.

This flexibility however comes at a cost: the explicit information (as exported by the mappings – see later) is not all the information available and hence reasoning tasks to support the extraction of such information is needed. This is why we need to carefully choose the ontology language so as to support efficiently main reasoning tasks such as query answering and constraint satisfiability checking. Fortunately, technologies and research results exists to support this, and we review them in the next section.

In fact, data are concretely stored in the artifacts, gateways and event databases, and manipulated by processes defined in the Artifact Layer, while they can be accessed at a higher level of abstraction through the Semantic Layer. This approach follows the ontology-based data access and integration paradigm, where access to the information contained in multiple, possibly heterogeneous data sources is mediated by an ontology that provides a "global", semantic account to such information.

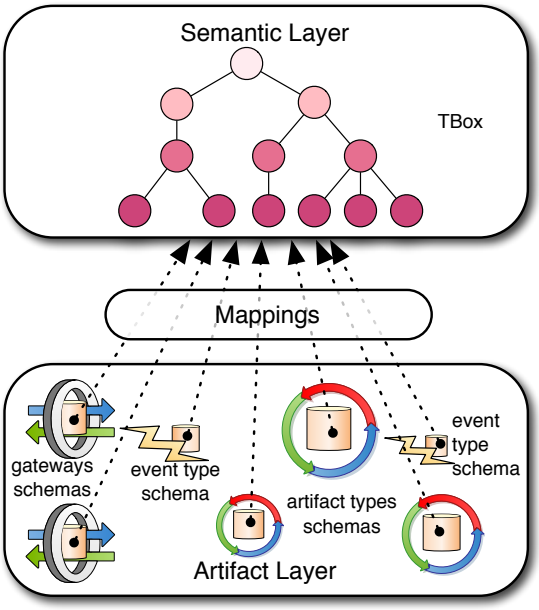


Figure 3 – Mappings as a glue between the Semantic and Artifact Layer

Mappings.

$\mathcal{M}$ is a mapping specification that captures the relationship between the Artifact Layer and the Semantic Layer. Such a relationship resembles a typical data integration setting, where artifacts, gateways and event databases are the data sources at the Artifact Layer, while the KAB's knowledge component constitutes the conceptual view of such data at the Semantic Layer. Therefore, as advocated by the recent literature on data integration, the connection between the Artifact Layer and the Semantic Layer is provided through suitable mappings [108, 88, 124], that express by means of mapping queries over the database schemas and the TBox how the data in the artifacts, the environment gateways and the events that constitute the Artifact Layer are related to (the virtual) data at the Semantic Layer (see Figure 3). Such mappings are then exploited in different ways: on the one hand, to satisfy information requests that are posed in terms of the Semantic Layer, and that require accessing data at the Artifact Layer; on the other hand, to understand tasks and processes that specify how to modify data stored

in artifact instances in terms of operations specified over the Semantic Layer [60, 50]. Since the mappings play a dominant role in this picture, the ontology language $\mathcal{L}$ used to describe the knowledge component in the Semantic Layer must be suitable for data access and integration, like those proposed in [41, 39, 7].

Such mappings may be of different nature. One typical example are the so called *Local As View (LAV)* mappings where the information content of the source is captured as a query over the virtual global view. For example, referring to Figure 4, we may have a TBox, expressed as a

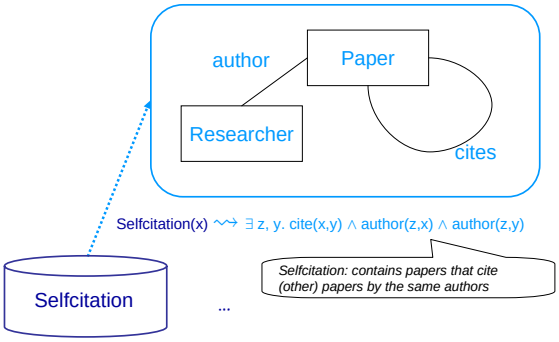
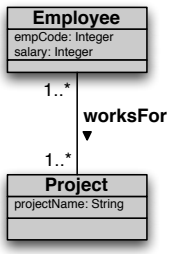


Figure 4 – LAV mappings

UML class diagram in the figure, talking about papers, researchers that are authors of papers, and papers that cite other papers. Then we have a source consisting of a table containing papers that we are selfcitations. A LAV mapping capturing this maps the tuple in the table to the results of a (virtual) query over the TBox that (virtually) retrieves the papers that cite a paper by the same author. The LAV mapping expresses that the content of the source is included in the result to the virtual query. More detail about these mappings will be given in the next section.



$D_1[\text{SSN}String, \text{PRNAME}String]$ Employees and Projects they work for
$D_2[\text{CODE}String, \text{SALARY}Int]$ Employee's code with salary
$D_3[\text{CODE}String, \text{SSN}String]$ Employee's code with SSN
...

m_1: `SELECT SSN, PrName FROM` D_1 $\rightsquigarrow$ EMPLOYEE(PERS(SSN)), PROJECT(PROJ(PRNAME)), PROJECTNAME(PROJ(PRNAME), PRNAME), WORKSFOR(PERS(SSN), PROJ(PRNAME))

m_2: `SELECT SSN, Salary FROM` D_2, D_3 `WHERE` D_2`.Code =` D_3`.Code` $\rightsquigarrow$ EMPLOYEE(PERS(SSN)), SALARY(PERS(SSN), SALARY)

Figure 5 – GAV mappings

Another typical kind of mappings are the so called *Global As View (LAV)* mappings where the information content of the concepts and relationships (roles) in the virtual global view are captured as queries over the sources. For example, referring to Figure 5, we may have a TBox, expressed as a UML class diagram in the figure, talking about employees working for projects. Data are contained in three data sources $D_1, D_2, D3$, and queries over such data sources are used to virtually populate the various classes (i.e. concepts), attributes and associations (i.e., roles) of the diagram (i.e., the TBox). Again more detail about these mappings will be given in the next section.

3.3 Action Component

While the knowledge component $\mathcal{K}$ focuses on the static, structural knowledge about the domain of interest as represented in the system, the *action component* Γ characterizes the dynamic aspects of such a domain of interest which are reflected in legal evolution of the data in the system. In its most general acception, Γ contains a set of *dynamic laws*, which implicitly define the behavioral boundaries that must be respected by processes and lifecycles defined in the Artifact Layer to reflect correctly the dynamics of the domain of interest. Consequently, the dynamics of the data manipulated by processes and artifact lifecycles, projected to the Semantic Layer through the application of the mappings, must obey to the laws imposed by the KAB action component.

To specify such dynamic laws one can consider sophisticated properties expressed in, for example, first-order variants of temporal logics used for specification and verification of systems such as LTL, CTL, μ-calculus, etc. [71, 139, 70, 90, 111, 69]. Examples of such formalisms are reported in the next section.

Observe that such properties are expressed at the semantic level, i.e., the logic that represents them has as "atomic formulas" arbitrary queries over the knowledge component, and on top of such queries temporal modalities of various forms are introduced. Another importan observation is that such properties have images in A^3M as static and dynamic constraints, involving data present in various artifacts and environment gateways. These constraints are again expressed in temporal logic though this ti gateways me in terms of the concrete data structures present in the artifact level.

From the operational point of view, the dynamic properties asserted in the Semantic Layer can be employed to check at design or deploy time (e.g., when a new artifact is introduced in the system) whether an artifact lifecycle or a new process specification are compliant with the KAB, once the mappings used to connect their data to the knowledge component are established. Also at runtime they can be used to monitor and track the execution of the system at the Artifact Layer from a more conceptual point of view, dealing directly with the domain of interest and not with the specific implementations of the artifacts themselves. Notice that in this way we may catch possible violations to desirable conditions both at the Semantic Layer and at the Artifact Layer (the latter through the images of semantics properties onto the Artifact Layer). Notice that such notion of compliance involves both the static and dynamic aspects of the KAB, i.e., the knowledge and the action components.

3.4 Semantic Snapshots and Traces

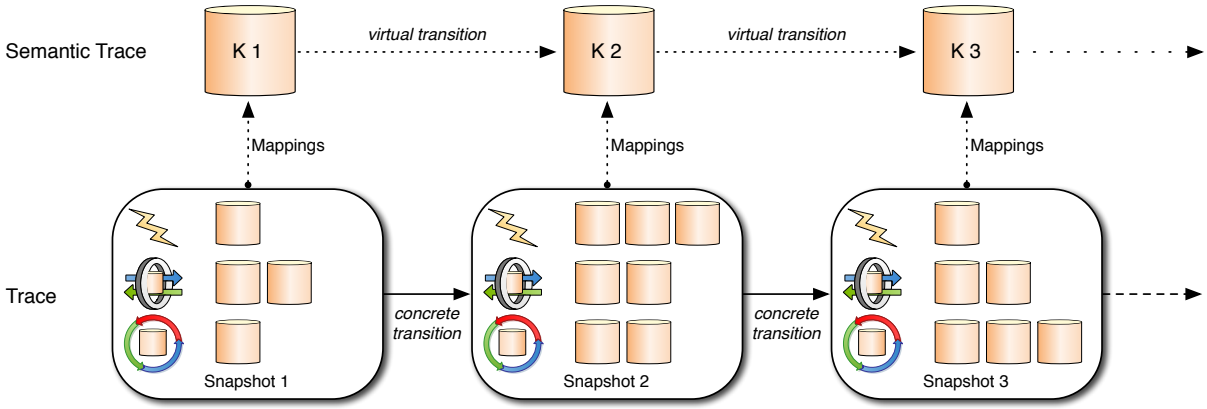


Figure 6 – A system trace and corresponding semantic trace.

As we have discussed for the knowledge component, the concrete data are maintained and manipulated at the Artifact Layer, while the Semantic Layer exports a virtual, conceptual view of them. Such projection is obtained through the applications of the mappings that interconnect

the database schemas present at the Artifact Layer to the KAB knowledge component. More specifically, these mappings are virtually applied to understand the transitions of the system occurring at the Artifact Layer in terms of "semantic" transitions: while the knowledge component (i.e., the ontology TBox) is supposed to remain fixed during the system's evolution, the extensions of concepts and roles described in the KAB are changed accordingly to the mappings (see Figure 6).

Technically, every snapshot characterizing a stable situation in a trace of the system is projected by the mappings to a sort of virtual ABox (see next section) at the Semantic Layer, which can be though of as a set of assertions on the extension of concepts and roles in the KAB's TBox. Notice that, typically, the TBox itself will remain unchanged during the execution of the system. Indeed the TBox captures intensional knowledge on concepts an relationships that is a the "schema level" and changes very rarely. The virtual ABox instead changes continuously to reflect the changes on the data at the artifact level. Such an ABox, complemented with the fixed TBox, characterizes the corresponding "semantic snapshot" of the current snapshot of system at the Artifact Layer. As a consequence, a trace at the Artifact Layer is virtually mapped onto a corresponding semantic trace at the Semantic Layer, where two consecutive ABoxes are connected through a "virtual transition" that results from an execution step performed at the Artifact Layer.[1] The same approach can be applied to understand at the Semantic Layer the transition systems formalizing processes running at the Artifact Layer. Notice, again, that the semantic traces/transition systems are required to obey to the static constraints imposed by the KAB knowledge component as well as to the dynamic laws belonging to the KAB action component.

3.5 Semantic Actions

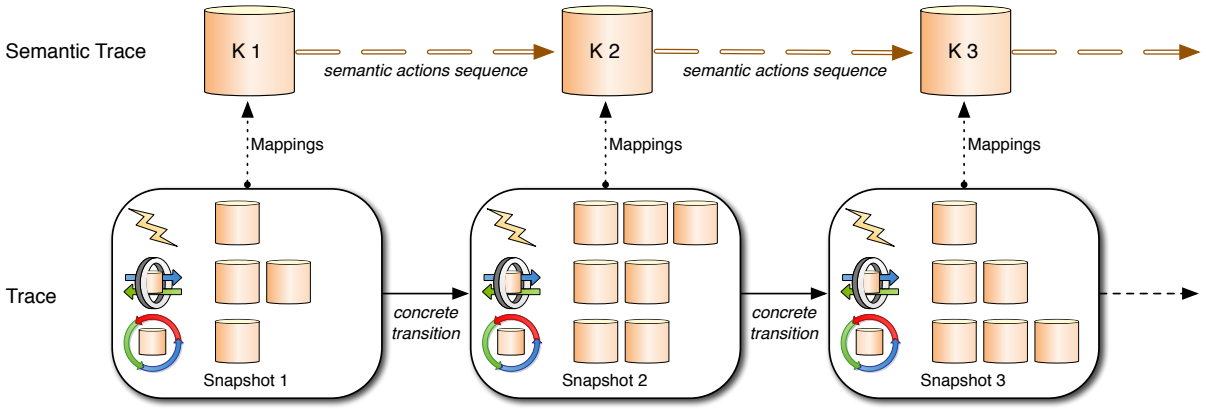


Figure 7 – Application of semantic actions

The dynamic laws contained in the KAB action component can range from declarative, coarse-grained properties to very detailed, fine-grained constraints that play the role of *semantic actions* expressed in terms of *pre* and *post*-conditions. In the latter case, semantic actions correspond to atomic actions that can be done in the domain of interest. Each semantic action corresponds to a conceptual building block used to describe an *atomic change* of the system at the Semantic Layer. Notice that there does not exist a direct correspondence between such actions of the action component and the real, concrete steps that determine a transition in the Artifact Layer, because the two layers work at different level of abstractions and, consequently, at different granularities. In particular, the atomic nature of semantic actions require them to be able to explain each step that leads the system to move from a snapshot at the Artifact

[1] Obviously the granularity of such steps can be chosen in several ways. For example in the GSM model of artifact systems, we are interested in mapping only the so called "macro-steps", which abstract form automated activities that all artifacts do to respond to a given external event, cf. Deliverable D1.1. See also the example at the end of this document.

Layer to a consequent successor snapshot: each such step must correspond to a *sequence* (or more generally, a composition) of semantic actions at the Semantic Layer, thus enforcing a sort of *serializability* (more generally, allowed compositions) of the actions forming the transition in terms of a sequence (composition) of semantic actions.

The situation is depicted in Figure 7, where each transition of the system at the Artifact Layer corresponds to a sequence of actions performed by the KAB. Notice that given two successive snapshot of the artifact system, we have two corresponding semantic snapshots at the Semantic Layer, such that from the first one it must be possible to reach the second one through a sequence of actions. If this is not the case then it means that, at the conceptual level, the domain of interest cannot evolve from the configuration corresponding to the first semantic snapshot to a configuration corresponding to the second one; in this case, the transition at the Artifact Layer is illegal and must be blocked.

3.6 Meta Information on the Artifact-Based System

As we have discussed, the main purpose of the knowledge component is to conceptually describe the important aspects of the domain of interest. Hence, in general the knowledge component abstracts away from how this information is organized in the Artifact Layer: artifacts, gateways and events may not be explicitly represented in the knowledge component, and only mentioned in the mappings to properly "feed" the Semantic Layer with the information maintained at the Artifact Layer. However, this by no means limits the possibility of modeling artifacts, gateways, events and their relationships at the Semantic Layer. In fact, all these entities can be seamlessly introduced into the KAB $\mathcal{KA}$, described in the knowledge component $\mathcal{K}$, and linked to the Artifact Layer through mappings. For example, the knowledge component could contain a hierarchy of relationship stereotypes, used to give a semantical account to the relationship stereotypes used in the Artifact Layer.

In this respect, the Semantic Layer can support a high level specification of the artifacts, and more in general employed to represent, query and govern meta information characterizing the entities contained in the Artifact Layer. The designer of the Semantic Layer can freely choose whether to incorporate such meta information into the knowledge component or not. Notice that the Semantic Layer provides support for even more sophisticated forms of modeling patterns; for example, Section 4.8 shows how to accommodate temporal information and, in turn, logs storing the evolution of an Artifact Framework at the Semantic Layer.

4 ACSI Semantic Layer: Technologies

In this section we collect all information needed on the technologies upon which the Semantic Layer can be formalized in detail and realized in practice. Specifically we introduce:

- Ontologies, Description Logics, and Ontology Based Data Access and Integration, to realize the TBox and the reasoning services over the KAB knowledge component
- Mappings, for linking the KAB to the data in the artifact layer.
- First-order variant of temporal logics, needed for realizing the KAB action base
- Semantic Log, that show various option on how one can log the system conceptually at the semantic layer.

The notions introduced here will be at the base of the further development of the ACSI Interoperation Hub in the other workpackages, and in particular in WP2.

4.1 Ontologies

Originally introduced by Parmenides and coupled with the study of Logics by Aristotle, *Ontologies* represent the philosophical study of *existence*, *reality* and the *nature of being*, as well as the study of all basic *categories of being*, and the *relations* among them. Since the mid-1970s, when researchers in the field of Artificial Intelligence (AI) recognized that capturing knowledge is the key to build large and powerful AI systems and that it was possible to create new ontologies as computational models enabling certain kinds of automated reasoning, the AI community began to use the term *ontology* (with lower case letter) to refer both to a theory of a modeled world and to a component of knowledge systems. So, drawing inspiration from philosophical *Ontologies*, since the mid-1980s researchers in Computer Science started viewing *computational ontologies* as a kind of applied philosophy, whose practical applications, are nowadays becoming more and more popular in various fields, ranging from the Semantic Web [91, 95], in which ontologies are seen as a key mechanisms to describe the semantics of information at various sites, to the Information and Data Integration scenario [108, 118, 46, 81], in which an ontology can be used as reconciled view over a set of data sources [62, 124, 38, 92].

Today the term ontology has come to refer to a wide range of formal representations, which makes it difficult to find a precise definition of what ontologies are. What they do, however, is to provide a formal conceptualization of a domain of interest, supporting human or machines to share some common knowledge in a structured way, and providing a premium mechanism through which services, operating in a Web context, can be accessed by human users and by other services. According to Gruber [83, 82], **an ontology is a formal, explicit specification of a shared conceptualization of a domain of interest**.

- A *conceptualization* is an abstract representation of some aspect of the world, which is of interest according to the users of the ontology.
- *Formal* means that the specification is encoded in a precisely defined language, whose properties are well known and understood. Usually this means that the languages used to specify the ontology are logic-based languages, such as those used in the Knowledge Representation and Artificial Intelligence communities.
- The term *explicit* refers to the constructs used in the specification, which must be explicitly defined, and to the fact that all users of the ontology, who share the information of interest and the ontology itself, must totally agree on them.
- *Shared* means that the ontology is meant to be shared across several people, applications, communities, and organizations.

As for the *knowledge* encoded in ontologies, according to Gruber [83, 82] it is mainly formalized using five kinds of components:

1. *concepts*, which represent sets of objects sharing some common properties within the domain of interest;
2. *relations*, which represent relationships among concepts, by means of the notion of mathematical relation;
3. *functions*, which are functional relations;
4. *individuals (or instances)*, which are individual objects in the domain of interest;
5. *axioms (or assertions)*, which are sentences that are always true and are used to enforce suitable properties of classes, relations, and individuals.

In other words, ontologies allow the key concepts and terms that are relevant to a given domain of interest to be identified and defined in an unambiguous way. Moreover, they facilitate the integration of different perspectives, while capturing key distinctions in a given perspective, improving the communication and cooperation of people or services within a single organization and across several organizations. Notice that, in order to reach this aim, it is fundamental that the language used to express ontologies might be a formal and machine-processable language.

4.2 Description Logics

An ontology, as a conceptualization of a domain of interest, provides the mechanisms for modeling the domain and reasoning upon it, and has to be represented in terms of a well-defined language. Description Logics (DLs) [9] are logics specifically designed to represent structured knowledge and to reason upon it, and as such are perfectly suited as languages for representing ontologies. Given a representation of the domain of interest, an ontology-based system should provide well-founded methods for reasoning upon it, i.e., for analyzing the representation, and drawing interesting conclusions about it. DLs, being logics, are equipped with reasoning methods, and DL-based systems provide reasoning algorithms and working implementations for them. This explains why variants of DLs are providing now the underpinning for the ontology languages promoted by the W3C, namely the standard Web Ontology Language OWL[2] and its variants (called *profiles*), which have been recently standardized by the W3C in their second edition, OWL 2.

DLs stem from the effort started in the mid 80s to provide a formal basis, grounded in logic, to formalisms for the structured representation of knowledge that were popular at that time, notably Semantic Networks and Frames [113, 26], that typically relied on graphical or network-like representation mechanisms. The fundamental work by Brachman and Levesque [24], initiated this effort, by showing on the one hand that the full power of First-Order Logic is not required to capture the most common representation elements, and on the other hand that the computational complexity of inference is highly sensitive to the expressive power of the KR language. Research in DLs up to our days can be seen as the systematic and exhaustive exploration of the corresponding tradeoff between expressiveness and efficiency of the various inference tasks associated to KR.

DLs are based on the idea that the knowledge in the domain to represent should be structured by grouping into classes objects of interest that have properties in common, and explicitly representing those properties through the relevant relationships holding among such classes. *Concepts* denote classes of objects, and *roles* denote (typically binary) relations between objects. Both are constructed, starting from atomic concepts and roles, by making use of various constructs, and it is precisely the set of allowed constructs that characterizes the (concept) language underlying a DL.

[2] `http://www.w3.org/2007/OWL/`

The domain of interest is then represented by means of a DL *knowledge base* (KB), where a separation is made between general intensional knowledge and specific knowledge about individual objects in the modeled domain. The first kind of knowledge is maintained in what has been traditionally called a *TBox* (for "Terminological Box"), storing a set of universally quantified assertions that state general properties of concepts and roles. The latter kind of knowledge is represented in an *ABox* (for "Assertional Box"), constituted by assertions on individual objects, e.g., the one stating that an individual is an instance of a certain concept.

Several reasoning tasks can be carried out on a DL KB, where the basic form of reasoning involves computing the subsumption relation between two concept expressions, i.e., verifying whether one expression always denotes a subset of the objects denoted by another expression. More in general, one is interested in understanding how the various elements of a KB interact with each other in an often complex way, possibly leading to inconsistencies that need to be detected, or implying new knowledge that should be made explicit.

The above observations emphasize that a DL system is characterized by three aspects:

(*i*) the set of constructs constituting the language for building the concepts and the roles used in a KB;
(*ii*) the kind of assertions that may appear in the KB;
(*iii*) the inference mechanisms provided for reasoning on the knowledge bases expressible in the system.

The expressive power and the deductive capabilities of a DL system depend on the various choices and assumptions that the system adopts with regard to the above aspects.

4.3 Expressive Power and Efficiency of Reasoning

The first aspect above, i.e., the language for concepts and roles, has been the subject of an intensive research work started in the late 80s. Indeed, the initial results on the computational properties of DLs have been devised in a simplified setting where both the TBox and the ABox are empty [116, 128, 66]. The aim was to gain a clear understanding of the properties of the language constructs and their interaction, with the goal of singling out their impact on the complexity of reasoning. Gaining this insight by understanding the combinations of language constructs that are *difficult* to deal with, and devising general methods to cope with them, is essential for the design of inference procedures. It is important to understand that in this context, the notion of "difficult" has to be understood in a precise technical sense, and the declared aim of research in this area has been to study and understand the frontier between tractability (i.e., solvable by a polynomial time algorithm) and intractability of reasoning over concept expressions. The maximal combinations of constructs (among those most commonly used) that still allowed for polynomial time inference procedures were identified, which allowed to exactly characterize the tractability frontier [66]. It should be noted that the techniques and technical tools that were used to prove such results, namely tableaux-based algorithms, are still at the basis of the modern state of the art DL reasoning systems [114], such as Fact [93], Racer [86], and Pellet [131, 130].

The research on the tractability frontier for reasoning over concept expressions allowed to precisely understand the properties and interactions of the various DL constructs, and identify practically meaningful combinations that are computationally tractable. However, from the point of view of knowledge representation, where knowledge about a domain needs to be encoded, maintained, and reasoned upon, the assumption of dealing with concept expressions only, without considering a KB (i.e., a TBox and possibly an ABox) to which the concepts refer, is clearly unrealistic. Early successful DL KR systems, such as Classic [122], relied on a KB, but did not renounce to tractability by imposing syntactic restrictions on the use of concepts in definitions, essentially to ensure acyclicity (i.e., lack of mutual recursion). Under such an assumption, the concept definitions in a KB can be folded away, and hence reasoning over a KB can be reduced to reasoning over concept expressions only.

However, the assumption of acyclicity is strongly limiting the ability to represent real-world knowledge. These limitations became quite clear also in light of the tight connection between DLs and formalisms for the structured representation of information used in other contexts, such as databases and software engineering [51]. In the presence of cyclic KBs, reasoning becomes provably exponential (i.e, ExpTime-complete) already when the concept language contains rather simple constructs. As a consequence of such a result, research in DLs shifted from the exploration of the tractability border to an exploration of the decidability border. The aim has been to investigate how much the expressive power of language and knowledge base constructs could be further increased while maintaining decidability of reasoning, possibly with the same, already rather high, computational complexity of inference. The techniques used to prove decidability and complexity results for expressive variants of DLs range from exploiting the correspondence with modal and dynamic logics [127, 47], to automata-based techniques [141, 140, 43, 45, 35, 10], to tableaux-based techniques [8, 29, 96, 11, 97]. It is worth noticing that the latter techniques, though not computationally optimal, are amenable to easier implementations, and are at the basis of the current state-of-the-art reasoners for expressive DLs [114].

4.4 Ontologies in data intensive applications

Current reasoners for expressive DLs perform indeed well in practice, and show that even procedures that are exponential in the size of the KB might be acceptable under suitable conditions. However, such reasoners have not specifically been tailored to deal with large amounts of data (e.g., a large ABox). This is especially critical in those settings where ontologies are used as a high-level, conceptual view over data repositories, allowing users to access data item without the need to know how the data is actually organized and where it is stored. Typical scenarios for this that are becoming more and more popular are those of Information and Data Integration Systems [108, 118, 46, 81], the Semantic Web [91, 95], and ontology-based data access [62, 124, 38, 92].

In these scenarios, data are typically very large and dominate the intentional level of the ontologies. Hence, while one could still accept reasoning that is exponential on the intentional part, it is mandatory that reasoning is polynomial (actually less – see later) in the data. If follows that, when measuring the computational complexity of reasoning, the most important parameter is the size of the data, i.e., one is interested in so-called *data complexity* [138]. Traditionally, research carried out in DLs has not paid much attention to the data complexity of reasoning, and only recently efficient management of large amounts of data [94, 54] has become a primary concern in ontology reasoning systems, and data-complexity has been studied explicitly [100, 39, 119, 106, 6, 7]. Unfortunately, research on the trade-off between expressive power and computational complexity of reasoning has shown that many DLs with efficient reasoning algorithms lack the modeling power required for capturing conceptual models and basic ontology languages. On the other hand, whenever the complexity of reasoning is exponential in the size of the instances (as for example for the expressive fragments of OWL and OW2, or in [44]), there is little hope for effective instance management.

A second fundamental requirement when one wants to access data using an ontology is the possibility to answer queries over an ontology that are more complex than the simple queries (i.e., concepts and roles) usually considered in DLs research. It turns out, however, that one cannot take the other extreme and adopt as a query language full SQL (corresponding to First-Order Logic queries), since due to the inherent incompleteness introduced by the presence of an ontology, query answering amounts to logical inference, which is undecidable for First-Order Logic. Hence, a good trade-off regarding the query language to use can be found by considering those query languages that have been advocated in databases in those settings where incompleteness of information is present [137], such as data integration [108] and data exchange [104, 110]. There, the query language of choice are *conjunctive queries*, corresponding to the select-project-join fragment of SQL, and unions thereof, which are also the kinds of queries that are best supported by commercial database management systems.

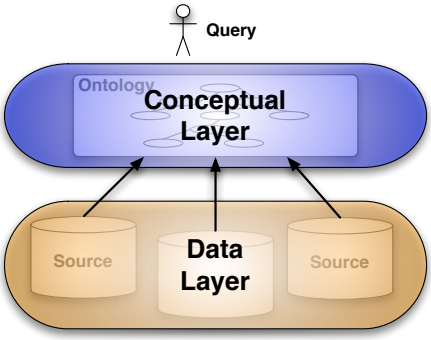


Figure 8 – Ontology-based Data Access

From the above discussion, it turns out that, for those contexts where ontologies are used to access large amounts of data, a suitable DL should be used, specifically tailored to capture all those constructs that are used typically in conceptual modeling, while keeping query answering efficient. Specifically, efficiency should be achieved by delegating data storage and query answering to a relational data management systems (RDBMS), which is the only technology that is currently available to deal with complex queries over large amounts of data. The chosen DL should include the main modeling features of conceptual models, which are also at the basis of most ontology languages. These features include cyclic assertions, ISA and disjointness of concepts and roles, inverses on roles, role typing, mandatory participation to roles, functional restrictions of roles, and a mechanisms for identifying instances of concepts. Also, the query language should go beyond the expressive capabilities of concept expressions in DLs, and allow for expressing conjunctive queries and unions thereof.

It is worth noting that when data management is delegated to an external RDBMS, a suitable mechanism as to be defined to couple (the intensional level of) the ontology, in fact a TBox, to the database representing the extensional level. This becomes even more crucial when the underlying database is not simply a faithful representation of the ABox of the ontology, but it is an autonomous relational database, over which a domain ontology has been superimposed to ease the access to its data. This may happen for several reasons. For example, databases may have undergone several manipulations during the years, often for optimizing applications using them, and may have lost their original design. They may have been distributed or replicated without a coherent design, so that the information turns out to be dispersed over several independent (maybe heterogeneous) data sources, and source data tend to be redundant and mutually inconsistent. In this context, *mappings* become crucial. Roughly speaking, in a mapping assertion a view (i.e., a query) over the ontology is put in correspondence with a view over the source database, with aim of specifying how instances of the ontology can be constructed starting from the data retrieved by querying the database. We call this context *Ontology-Based Data Access* (OBDA) [124, 38, 37] (cf. Figure 8).

OBDA is particularly relevant for ACSI, where the ontology specified in the Knowledge and Action Base is connected with data managed in the underlying artifacts through suitable mappings. Therefore, we are particular interested in looking in more depth the aspects we have mentioned in this section, i.e., analyzing data-oriented DLs which enables delegating management of the data layer to an RDBMS, but at the same time offer the expressive power of basic ontology languages, and defining services for data extraction, i.e., expressive query answering, over such DLs. Furthermore, we want to look at various mechanisms for specifying mappings and thus we need to draw from the field of data integration, then adapting to the presence of ontologies the forms of mappings used for integrating data. All this is the subject of the next subsections.

4.5 The Description Logic *DL-Lite*$_{\mathcal{A},id}$

We now introduce formally syntax and semantics of DLs, and we do so for *DL-Lite*$_{\mathcal{A},id}$, a specific DL of the *DL-Lite* family [41, 39], that is also equipped with identification constraints [42]. *DL-Lite*$_{\mathcal{A},id}$ is able to capture the most significant features of popular conceptual modeling formalisms we mentioned in the previous section, nevertheless query answering of complex queries (i.e., conjunctive queries) can be managed efficiently by relying on relational database technology. For this reason, *DL-Lite* is particularly suited for dealing with large databases managed in mass memory through an RDBMS. Furthermore, *DL-Lite*$_{\mathcal{A},id}$ is at the basis of OWL 2 QL, one of the three profiles of OWL 2 that have been recently standardized by the World-Wide-Web Consortium (W3C). The *OWL 2 profiles*[3] are fragments of the full OWL 2 language that have been designed and standardized for specific application requirements. According to (the current version of) the official W3C profiles document, "OWL 2 QL includes most of the main features of conceptual models such as UML class diagrams and ER diagrams. [It] is aimed at applications that use very large volumes of instance data, and where query answering is the most important reasoning task. In OWL 2 QL, conjunctive query answering can be implemented using conventional relational database systems."

Syntax. As mentioned, in Description Logics [9] (DLs) the domain of interest is modeled by means of *concepts*, which denote classes of objects, and *roles* (i.e., binary relationships), which denote binary relations between objects. In addition, *DL-Lite*$_{\mathcal{A},id}$ distinguishes concepts from *value-domains*, which denote sets of (data) values, and roles from *attributes*, which denote binary relations between objects and values. We now define formally syntax and semantics of expressions in our logic.

Like in any other logic, *DL-Lite*$_{\mathcal{A},id}$ expressions are built over an alphabet. In our case, the alphabet comprises symbols for atomic concepts, value-domains, atomic roles, atomic attributes, and constants. In particular, for the set of constants Γ, we pose $\Gamma = \Gamma_O \cup \Gamma_V$, where Γ_O is the set of constants denoting objets and Γ_V is the set of constants denoting values, and $\Gamma_O \cap \Gamma_V = \emptyset$. Concepts, roles, attributes, and value-domains in this DL are formed according to the following syntax:

$$
\begin{array}{lcl@{\qquad\qquad}lcl}
B & \longrightarrow & A \mid \exists Q \mid \delta(U) & E & \longrightarrow & \rho(U) \\
C & \longrightarrow & B \mid \neg B & F & \longrightarrow & \top_D \mid T_1 \mid \cdots \mid T_n \\
Q & \longrightarrow & P \mid P^- & V & \longrightarrow & U \mid \neg U \\
R & \longrightarrow & Q \mid \neg Q & & & \\
\end{array}
$$

In such rules, A, P, and P^- respectively denote an *atomic concept*, an *atomic role*, and the *inverse of an atomic role*, Q and R respectively denote a *basic role* and an *arbitrary role*, whereas B denotes a *basic concept*, C an *arbitrary concept*, U an *atomic attribute*, V an *arbitrary attribute*, E a *basic value-domain*, and F an *arbitrary value-domain*. Furthermore, $\delta(U)$ denotes the *domain* of U, i.e., the set of objects that U relates to values; $\rho(U)$ denotes the *range* of U, i.e., the set of values that U relates to objects; $\top_D$ is the universal value-domain; $T_1, \ldots, T_n$ are n pairwise disjoint unbounded value-domains, corresponding to RDF data types, such as `xsd:string`, `xsd:integer`, etc.

Like in any DL, a *DL-Lite*$_{\mathcal{A},id}$ *ontology*, or knowledge base (KB), is constituted by two components:

- a *TBox* (where the 'T' stands for *terminological*), which is a finite set of *intensional assertions*, and
- an *ABox* (where the 'A' stands for *assertional*), which is a finite set of *extensional* (or, *membership*) *assertions*.

We now specify formally the form of a *DL-Lite*$_{\mathcal{A},id}$ TBox and ABox.

[3] `http://www.w3.org/TR/owl2-profiles/`

In a *DL-Lite*$_{\mathcal{A},id}$ TBox, assertions have the forms

$$\begin{array}{cccc} B \sqsubseteq C & Q \sqsubseteq R & U \sqsubseteq V & E \sqsubseteq F \\ (\mathsf{funct}\ Q) & (\mathsf{funct}\ U) & (\mathsf{id}\ B\ \pi_1, \ldots, \pi_n) & \end{array}$$

The assertions above, from left to right, respectively denote inclusions between concepts, roles, attributes, and value-domains, (global) functionality on roles and on attributes, and identification constraints. Intuitively, the inclusion $B \sqsubseteq C$ expresses that all instances of concept B are also instances of concept C (the other inclusions express analogous properties). Notice that we allow negation $\neg$ only in the right-hand side of concept, role and attribute inclusions, which is sufficient to model disjointness properties between concepts, roles, or attributes, respectively. Global functionalities allows instead for specifying that roles or attributes are indeed functions.

Something more has to be said for identification assertions, which are mechanisms for identifying instances of concepts. In an identification assertion (also called identification constraint) $(\mathsf{id}\ B\ \pi_1, \ldots, \pi_n)$, B is a basic concept, $n \geq 1$, and $\pi_1, \ldots, \pi_n$ (called the *components* of the identifier) are paths. A *path* is an expression built according to the following syntax,

$$\pi \ \longrightarrow \ S \ \mid \ D? \ \mid \ \pi_1 \circ \pi_2 \tag{1}$$

where S denotes a basic role (i.e., an atomic role or the inverse of an atomic role), an atomic attribute, or the inverse of an atomic attribute, and $\pi_1 \circ \pi_2$ denotes the composition of the paths π_1 and π_2. Finally, D denotes an basic concept or a (basic or arbitrary) value domain, and the expression $D?$ is called a *test relation*, which represents the identity relation on instances of D. Test relations are used in all those cases in which one wants to impose that a path involves instances of a certain concept. For example, *HAS-CHILD* $\circ$ *Woman*? is the path connecting someone with his/her daughters. A path π denotes a complex property for the instances of concepts: given an object o, every object that is reachable from o by means of π is called a *π-filler* for o. Note that for a certain o there may be several distinct π-fillers, or no π-fillers at all. If π is a path, the length of π, denoted $\mathit{length}(\pi)$, is 0 if π has the form $D?$, is 1 if π has the form S, and is $\mathit{length}(\pi_1) + \mathit{length}(\pi_2)$ if π has the form $\pi_1 \circ \pi_2$. In an identification assertion $(\mathsf{id}\ B\ \pi_1, \ldots, \pi_n)$ we have that $\mathit{length}(\pi_i) \geq 1$ for all $i \in \{1, \ldots, n\}$.

Intuitively, an identification assertion of the form above asserts that for any two different instances o, o' of B, there is at least one π_i such that o and o' differ in the set of their π_i-fillers. The identification assertion is called *local* if $\mathit{length}(\pi_i) = 1$ for at least one $i \in \{1, \ldots, n\}$. The term "local" emphasizes that at least one of the paths has length 1 and thus refers to a local property of B. For example, the identification assertion $(\mathsf{id}$ *Woman HAS-HUSBAND*$)$ says that a woman is identified by her husband, i.e., there are not two different women with the same husband, whereas the identification assertion $(\mathsf{id}$ *Man HAS-CHILD*$)$ says that a man is identified by his children, i.e., there are not two men with a child in common. We can also say that there are not two men with the same daughters by means of the identification $(\mathsf{id}$ *Man HAS-CHILD* $\circ$ *Woman*?$)$.

A *DL-Lite*$_{\mathcal{A},id}$ ABox is a finite set of assertions of the form $A(a)$, $P(a, b)$, and $U(a, v)$, where A is an atomic concept, P is an atomic role, U is an atomic attribute, a and b are constants, and v is a value.

We are now ready to define what a *DL-Lite*$_{\mathcal{A},id}$ KB is.

Definition 4.1 *A DL-Lite*$_{\mathcal{A},id}$ *KB $\mathcal{K}$ is a pair $\langle \mathcal{T}, \mathcal{A} \rangle$, where $\mathcal{T}$ is a DL-Lite*$_{\mathcal{A},id}$ *TBox, $\mathcal{A}$ is a DL-Lite*$_{\mathcal{A},id}$ *ABox, and the following conditions are satisfied:*

1. *for each atomic role P, if either* $(\mathsf{funct}\ P)$ *or* $(\mathsf{funct}\ P^-)$ *occur in $\mathcal{T}$, then $\mathcal{T}$ does not contain assertions of the form $Q \sqsubseteq P$ or $Q \sqsubseteq P^-$, where Q is a basic role;*
2. *for each atomic attribute U, if* $(\mathsf{funct}\ U)$ *occurs in $\mathcal{T}$, then $\mathcal{T}$ does not contain assertions of the form $U' \sqsubseteq U$, where U' is an atomic attribute;*
3. *all concepts identified in $\mathcal{T}$ are basic concepts, i.e., in each IdC* $(\mathsf{id}\ C\ \pi_1, \ldots, \pi_n)$ *of $\mathcal{T}$, the concept C is of the form A, $\exists Q$, or $\delta(U)$;*

$$
\begin{array}{rcl@{\qquad}rcl}
A^{\mathcal{I}} & \subseteq & \Delta_O^{\mathcal{I}} & T_i^{\mathcal{I}} & = & val(T_i) \\
(\exists Q)^{\mathcal{I}} & = & \{\ o \mid \exists o' \boldsymbol{.}\ (o,o') \in Q^{\mathcal{I}}\ \} & P^{\mathcal{I}} & \subseteq & \Delta_O^{\mathcal{I}} \times \Delta_O^{\mathcal{I}} \\
(\delta(U))^{\mathcal{I}} & = & \{\ o \mid \exists v \boldsymbol{.}\ (o,v) \in U^{\mathcal{I}}\ \} & (P^-)^{\mathcal{I}} & = & \{\ (o,o') \mid (o',o) \in P^{\mathcal{I}}\ \} \\
(\neg B)^{\mathcal{I}} & = & \Delta_O^{\mathcal{I}} \setminus B^{\mathcal{I}} & (\neg Q)^{\mathcal{I}} & = & (\Delta_O^{\mathcal{I}} \times \Delta_O^{\mathcal{I}}) \setminus Q^{\mathcal{I}} \\
(\rho(U))^{\mathcal{I}} & = & \{\ v \mid \exists o \boldsymbol{.}\ (o,v) \in U^{\mathcal{I}}\ \} & U^{\mathcal{I}} & \subseteq & \Delta_O^{\mathcal{I}} \times \Delta_V^{\mathcal{I}} \\
\top_d^{\mathcal{I}} & = & \Delta_V^{\mathcal{I}} & (\neg U)^{\mathcal{I}} & = & (\Delta_O^{\mathcal{I}} \times \Delta_V^{\mathcal{I}}) \setminus U^{\mathcal{I}}
\end{array}
$$

Figure 9 – Semantics of *DL-Lite*$_{\mathcal{A},id}$ expressions.

4. *all concepts or value-domains appearing in the test relations in $\mathcal{T}$ are of the form A, $\exists Q$, $\delta(U)$, $\rho(U)$, $\top_D$, T_1, ..., or T_n;*

5. *for each IdC α in $\mathcal{T}$, every role or attribute that occurs (in either direct or inverse direction) in a path of α is not specialized in $\mathcal{T}'$, i.e., it does not appear in the right-hand side of assertions of the form $Q \sqsubseteq Q'$ or $U \sqsubseteq U'$.*

Intuitively, the conditions stated at points (1-2) (resp., (5)) say that, in *DL-Lite*$_{\mathcal{A},id}$ TBoxes, roles and attributes occurring in functionality assertions (resp., in paths of IdCs) cannot be specialized. All the above conditions are crucial for the tractability of reasoning in our logic.

Semantics. Following the classical approach in DLs, the semantics of *DL-Lite*$_{\mathcal{A},id}$ is given in terms of First-Order Logic interpretations. All such interpretations agree on the semantics assigned to each value-domain T_i and to each constant in Γ_V. In particular, each valued-domain T_i is interpreted as the set $val(T_i)$ of values of the corresponding RDF data type, and each constant $c_i \in \Gamma_V$ is interpreted as one specific value, denoted $val(c_i)$, in $val(T_i)$. Note that, since the data types T_i are pairwise disjoint, we have that $val(T_i) \cap val(T_j) = \emptyset$, for $i \neq j$.

Then, an *interpretation* in *DL-Lite*$_{\mathcal{A},id}$ is a pair $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$, where

- $\Delta^{\mathcal{I}}$ is the interpretation domain, which is the disjoint union of two non-empty sets: $\Delta_O^{\mathcal{I}}$, called the *domain of objects*, and $\Delta_V^{\mathcal{I}}$, called the *domain of values*. In turn, $\Delta_V^{\mathcal{I}}$ is the union of $val(T_1), \ldots, val(T_n)$.

- $\cdot^{\mathcal{I}}$ is the *interpretation function*, i.e., a function that assigns an element of $\Delta^{\mathcal{I}}$ to each constant in Γ, a subset of $\Delta^{\mathcal{I}}$ to each concept and value-domain, and a subset of $\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$ to each role and attribute, in such a way that the following holds:
  - for each $c \in \Gamma_V$, $c^{\mathcal{I}} = val(c)$,
  - for each $d \in \Gamma_O$, $d^{\mathcal{I}} \in \Delta_O^{\mathcal{I}}$,
  - for each $a_1, a_2 \in \Gamma$, $a_1 \neq a_2$ implies $a_1^{\mathcal{I}} \neq a_2^{\mathcal{I}}$, and
  - the conditions shown in Figure 9 are satisfied.

Note that the above definition implies that different constants are interpreted differently in the domain, i.e., *DL-Lite*$_{\mathcal{A},id}$ adopts the so-called *unique name assumption* (UNA).

Then, we say that a *DL-Lite*$_{\mathcal{A},id}$ interpretation satisfies a *DL-Lite*$_{\mathcal{A},id}$ KB $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$ if it satisfies all assertions α in $\mathcal{K}$ (either an intensional assertion or a membership assertion), denoted $\mathcal{I} \models \alpha$. More precisely, we have that:

- An interpretation $\mathcal{I}$ satisfies a concept (resp., value-domain, role, attribute) inclusion assertion

$$
\begin{array}{lcll}
B & \sqsubseteq & C, & \text{if} \quad B^{\mathcal{I}} \subseteq C^{\mathcal{I}}; \\
E & \sqsubseteq & F, & \text{if} \quad E^{\mathcal{I}} \subseteq F^{\mathcal{I}}; \\
Q & \sqsubseteq & R, & \text{if} \quad Q^{\mathcal{I}} \subseteq R^{\mathcal{I}}; \\
U & \sqsubseteq & V, & \text{if} \quad U^{\mathcal{I}} \subseteq V^{\mathcal{I}}.
\end{array}
$$

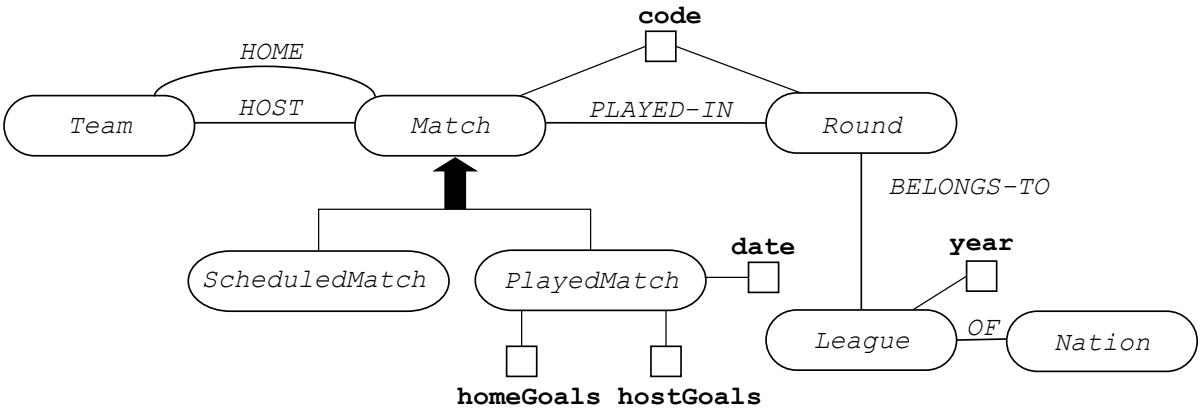


Figure 10 – Diagrammatic representation of the football championship ontology.

- An interpretation $\mathcal{I}$ satisfies a role functionality assertion (funct Q), if for each $o_1, o_2, o_3 \in \Delta_O^{\mathcal{I}}$

$$(o_1, o_2) \in Q^{\mathcal{I}} \text{ and } (o_1, o_3) \in Q^{\mathcal{I}} \quad \text{implies} \quad o_2 = o_3.$$

- An interpretation $\mathcal{I}$ satisfies an attribute functionality assertion (funct U), if for each $o \in \Delta_O^{\mathcal{I}}$ and $v_1, v_2 \in \Delta_V^{\mathcal{I}}$

$$(o, v_1) \in U^{\mathcal{I}} \text{ and } (o, v_2) \in U^{\mathcal{I}} \quad \text{implies} \quad v_1 = v_2.$$

- In order to define the semantics of identification assertions, we first define the semantics of paths. The extension $\pi^{\mathcal{I}}$ of a path π in an interpretation $\mathcal{I}$ is defined as follows:
  - if $\pi = S$, then $\pi^{\mathcal{I}} = S^{\mathcal{I}}$,
  - if $\pi = D?$, then $\pi^{\mathcal{I}} = \{(o, o) \mid o \in D^{\mathcal{I}}\}$,
  - if $\pi = \pi_1 \circ \pi_2$, then $\pi^{\mathcal{I}} = \pi_1^{\mathcal{I}} \circ \pi_2^{\mathcal{I}}$, where $\circ$ denotes the composition operator on relations.

  As a notation, we write $\pi^{\mathcal{I}}(o)$ to denote the set of π-fillers for o in $\mathcal{I}$, i.e., $\pi^{\mathcal{I}}(o) = \{o' \mid (o, o') \in \pi^{\mathcal{I}}\}$.

  Then, an interpretation $\mathcal{I}$ satisfies an identification assertion (id B $\pi_1, \ldots, \pi_n$) if for all $o, o' \in B^{\mathcal{I}}$, $\pi_1^{\mathcal{I}}(o) \cap \pi_1^{\mathcal{I}}(o') \neq \emptyset \wedge \cdots \wedge \pi_n^{\mathcal{I}}(o) \cap \pi_n^{\mathcal{I}}(o') \neq \emptyset$ implies $o = o'$. Observe that this definition is coherent with the intuitive reading of identification assertions discussed above, in particular by sanctioning that two different instances o, o' of B differ in the set of their π_i-fillers when such sets are disjoint.

- An interpretation $\mathcal{I}$ satisfies a membership assertion

$$\begin{array}{lll} A(a), & \text{if} & a^{\mathcal{I}} \in A^{\mathcal{I}}; \\ P(a_1, a_2), & \text{if} & (a_1^{\mathcal{I}}, a_2^{\mathcal{I}}) \in P^{\mathcal{I}}; \\ U(a, c), & \text{if} & (a^{\mathcal{I}}, c^{\mathcal{I}}) \in U^{\mathcal{I}}. \end{array}$$

An interpretation $\mathcal{I}$ that satisfies all assertions in a *DL-Lite*$_{\mathcal{A},id}$ ontology $\mathcal{O}$ (resp., TBox $\mathcal{T}$, ABox $\mathcal{A}$), is a *model* for $\mathcal{O}$ (resp., TBox $\mathcal{T}$, ABox $\mathcal{A}$), written $\mathcal{I} \models \mathcal{O}$ (resp., $\mathcal{I} \models \mathcal{T}$, $\mathcal{I} \models \mathcal{A}$). The semantics of a *DL-Lite*$_{\mathcal{A},id}$ ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$ is the set of all *models* of $\mathcal{O}$.

Example 4.1 We conclude the presentation of *DL-Lite*$_{\mathcal{A},id}$ with an example in which we present a *DL-Lite*$_{\mathcal{A},id}$ ontology modeling the annual national football championships in Europe, where the championship for a specific nation is called *league* (e.g., the Spanish Liga). A league is structured in terms of a set of *rounds*. Every round contains a set of *matches*, each one characterized by one *home team* and one *host team*. We distinguish between scheduled matches, i.e., matches

Inclusion assertions					
$League$	$\sqsubseteq$	$\exists OF$	$PlayedMatch$	$\sqsubseteq$	$Match$
$\exists OF$	$\sqsubseteq$	$League$	$ScheduledMatch$	$\sqsubseteq$	$Match$
$\exists OF^-$	$\sqsubseteq$	$Nation$	$PlayedMatch$	$\sqsubseteq$	$\neg ScheduledMatch$
$Round$	$\sqsubseteq$	$\exists BELONGS\text{-}TO$	$Match$	$\sqsubseteq$	$\neg Round$
$\exists BELONGS\text{-}TO$	$\sqsubseteq$	$Round$	$League$	$\sqsubseteq$	$\delta(\mathbf{year})$
$\exists BELONGS\text{-}TO^-$	$\sqsubseteq$	$League$	$Match$	$\sqsubseteq$	$\delta(\mathbf{code})$
$Match$	$\sqsubseteq$	$\exists PLAYED\text{-}IN$	$Round$	$\sqsubseteq$	$\delta(\mathbf{code})$
$\exists PLAYED\text{-}IN$	$\sqsubseteq$	$Match$	$PlayedMatch$	$\sqsubseteq$	$\delta(\mathbf{date})$
$\exists PLAYED\text{-}IN^-$	$\sqsubseteq$	$Round$	$PlayedMatch$	$\sqsubseteq$	$\delta(\mathbf{homeGoals})$
$Match$	$\sqsubseteq$	$\exists HOME$	$PlayedMatch$	$\sqsubseteq$	$\delta(\mathbf{hostGoals})$
$\exists HOME$	$\sqsubseteq$	$Match$	$\rho(\mathbf{date})$	$\sqsubseteq$	`xsd:date`
$\exists HOME^-$	$\sqsubseteq$	$Team$	$\rho(\mathbf{homeGoals})$	$\sqsubseteq$	`xsd:nonNegativeInteger`
$Match$	$\sqsubseteq$	$\exists HOST$	$\rho(\mathbf{hostGoals})$	$\sqsubseteq$	`xsd:nonNegativeInteger`
$\exists HOST$	$\sqsubseteq$	$Match$	$\rho(\mathbf{code})$	$\sqsubseteq$	`xsd:string`
$\exists HOST^-$	$\sqsubseteq$	$Team$	$\rho(\mathbf{year})$	$\sqsubseteq$	`xsd:positiveInteger`

Functionality assertions			
(funct *OF*)	(funct *HOME*)	(funct **year**)	(funct **homeGoals**)
(funct *BELONGS-TO*)	(funct *HOST*)	(funct **code**)	(funct **hostGoals**)
(funct *PLAYED-IN*)		(funct **date**)	

Identification assertions	
1. (id *League OF*, **year**)	6. (id *PlayedMatch* **date**, *HOST*)
2. (id *Round BELONGS-TO*, **code**)	7. (id *PlayedMatch* **date**, *HOME*)
3. (id *Match PLAYED-IN*, **code**)	8. (id *League* **year**, $BELONGS\text{-}TO^- \circ PLAYED\text{-}IN^- \circ HOME$)
4. (id *Match HOME*, *PLAYED-IN*)	9. (id *League* **year**, $BELONGS\text{-}TO^- \circ PLAYED\text{-}IN^- \circ HOST$)
5. (id *Match HOST*, *PLAYED-IN*)	10. (id *Match HOME*, *HOST*, $PLAYED\text{-}IN \circ BELONGS\text{-}TO \circ$ **year**)

Figure 11 – The *DL-Lite*$_{\mathcal{A},id}$ TBox $\mathcal{T}_{fbc}$ for the football championship example.

that have still to be played, and played matches. Obviously, a match falls in exactly one of these two categories.

In Figure 10, we show a schematic representation of (a portion of) the intensional part of the ontology for the football championship domain. In this figure, the black arrow represents a partition of one concept into a set of sub-concepts. We have not represented explicitly in the figure the pairwise disjointness of the concepts *Team*, *Match*, *Round*, *League*, and *Nation*, which intuitively holds in the modeled domain. In Figure 11, a *DL-Lite*$_{\mathcal{A},id}$ TBox $\mathcal{T}_{fbc}$ is shown that captures (most of) the above aspects. In our examples, we use the *CapitalizedItalics* font to denote atomic concepts, the *ALL-CAPITALS-ITALICS* font to denote atomic roles, the `typewriter` font to denote value-domains, and the **boldface** font to denote atomic attributes. Regarding the pairwise disjointness of the various concepts, we have represented by means of negative inclusion assertions only the disjointness between *PlayedMatch* and *ScheduledMatch* and the one between *Match* and *Round*. By virtue of the characteristics of *DL-Lite*$_{\mathcal{A},id}$, we can explicitly consider also attributes of concepts and the fact that they are used for identification. In particular, we assume that when a scheduled match takes place, it is played in a specific date, and that for every match that has been played, the number of goals scored by the home team and by the host team are given. Note that different matches scheduled for the same round can be played in different dates. Also, we want to distinguish football championships on the basis of the nation and the year in which a championship takes place (e.g., the 2009 Italian Liga). We also assume that both matches and rounds have codes. The identification assertions model the following aspects:

1. No nation has two leagues in the same year.
2. Within a league, the code associated to a round is unique.
3. Every match is identified by its code within its round.
4. A team is the home team of at most one match per round.
5. As above for the host team.

Concept and role membership assertions		
$League(it2009)$		$PLAYED\text{-}IN(m7RJ, r7)$
$Round(r7)$	$BELONGS\text{-}TO(r7, it2009)$	$PLAYED\text{-}IN(m8NT, r8)$
$Round(r8)$	$BELONGS\text{-}TO(r8, it2009)$	$PLAYED\text{-}IN(m8RM, r8)$
$PlayedMatch(m7RJ)$	$HOME(m7RJ, roma)$	$HOST(m7RJ, juventus)$
$Match(m8NT)$	$HOME(m8NT, napoli)$	$HOST(m8NT, torino)$
$Match(m8RM)$	$HOME(m8RM, roma)$	$HOST(m8RM, milan)$
$Team(roma)$	$Team(napoli)$	$Team(juventus)$
Attribute membership assertions		
code$(r7, \texttt{"7"})$	**code**$(m7RJ, \texttt{"RJ"})$	**date**$(m7RJ, \texttt{5/4/09})$
code$(r8, \texttt{"8"})$	**code**$(m8NT, \texttt{"NT"})$	**homeGoals**$(m7RJ, \texttt{3})$
	code$(m8RM, \texttt{"RM"})$	**hostGoals**$(m7RJ, \texttt{1})$

Figure 12 – The ABox $\mathcal{A}_{fbc}$ for the football championship example.

6. No home team participates in different played matches in the same date.
7. As above for the host team.
8. No home team plays in different leagues in the same year.
9. As above for the host team.
10. No pair (home team, host team) plays different matches in the same year.

Note that the *DL-Lite*$_{\mathcal{A},id}$ TBox in Figure 11 captures the ontology in Figure 10, except for the fact that the concept *Match* covers the concepts *ScheduledMatch* and *PlayedMatch*. In order to express such a condition, we would need to use disjunction in the right-hand-side of inclusion assertions, i.e.,

$$Match \sqsubseteq ScheduledMatch \sqcup PlayedMatch$$

where $\sqcup$ would be interpreted as set union. We have to renounce to the expressive power required to capture covering constraints (i.e., disjunction), if we want to preserve nice computational properties for reasoning over *DL-Lite*$_{\mathcal{A},id}$ ontologies.

An ABox, $\mathcal{A}_{fbc}$, associated to the TBox in Figure 11 is shown in Figure 12, where we have used the *slanted* font for constants in Γ_O and the `typeface` font for constants in Γ_V. For convenience of reading, we have chosen in the example names of the constants that indicate the properties of the objects that the constants represent.

We observe that the ontology $\mathcal{O}_{fbc} = \langle \mathcal{T}_{fbc}, \mathcal{A}_{fbc} \rangle$ is satisfiable. Indeed, the interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ shown in Figure 13 is a model of the ABox $\mathcal{A}_{fbc}$, where we have assumed that for each value constant $c \in \Gamma_V$, the corresponding value $val(c)$ is equal to c itself, hence $c^{\mathcal{I}} = val(c) = c$. Moreover, it is easy to see that every interpretation $\mathcal{I}$ has to satisfy the conditions shown in Figure 13 in order to be a model of $\mathcal{A}_{fbc}$. Furthermore, the following are necessary conditions for $\mathcal{I}$ to be also a model of the TBox $\mathcal{T}_{fbc}$, and hence of $\mathcal{O}_{fbc}$:

$it2009^{\mathcal{I}} \in (\exists OF)^{\mathcal{I}}$	to satisfy	$League \sqsubseteq \exists OF$,
$it2009^{\mathcal{I}} \in (\delta(\textbf{year}))^{\mathcal{I}}$	to satisfy	$League \sqsubseteq \delta(\textbf{year})$,
$m7RJ^{\mathcal{I}} \in Match^{\mathcal{I}}$	to satisfy	$PlayedMatch \sqsubseteq Match$,
$torino^{\mathcal{I}} \in Team^{\mathcal{I}}$	to satisfy	$\exists HOST^{-} \sqsubseteq Team$,
$milan^{\mathcal{I}} \in Team^{\mathcal{I}}$	to satisfy	$\exists HOST^{-} \sqsubseteq Team$.

Notice that, in order for an interpretation $\mathcal{I}$ to satisfy the condition specified in the first row above, there must be an object $o \in \Delta_O^{\mathcal{I}}$ such that $(it2009^{\mathcal{I}}, o) \in OF^{\mathcal{I}}$. According to the

inclusion assertion $\exists OF^{-} \sqsubseteq Nation$, such an object o must also belong to $Nation^{\mathcal{I}}$ (indeed, in our ontology, every league is of one nation). Similarly, the second row above derives from the property that every league must have a year.

We note that, besides satisfying the conditions discussed above, an interpretation $\mathcal{I}'$ may also add other elements to the interpretation of concepts, attributes, or roles specified by $\mathcal{I}$. For instance, the interpretation $\mathcal{I}'$ that adds to $\mathcal{I}$ the object

$$italy^{\mathcal{I}} \in Nation^{\mathcal{I}}$$

is still a model of the ontology $\mathcal{O}_{fbc}$.

Note, finally, that there exists no model of $\mathcal{O}_{fbc}$ such that *m7RJ* is interpreted as an instance of *ScheduledMatch*, since *m7RJ* has to be interpreted as an instance of *PlayedMatch*, and according to the inclusion assertion

$$PlayedMatch \sqsubseteq \neg ScheduledMatch,$$

the sets of played matches and of scheduled matches are disjoint. ∎

The above example clearly shows the difference between a database and an ontology. From a database point of view the ontology $\mathcal{O}_{fbc}$ discussed in the example might seem incorrect: for example, while the TBox $\mathcal{T}_{fbc}$ sanctions that every league has a year, there is no explicit year for *it2009* in the ABox $\mathcal{A}_{fbc}$. However, the ontology is not incorrect: the axiom stating that every league has a year simply specifies that in every model of $\mathcal{O}_{fbc}$ there will be a year for *it2009*, even if such a year is not known.

Queries over *DL-Lite*$_{\mathcal{A},id}$ Ontologies. We are interested in queries over ontologies expressed in *DL-Lite*$_{\mathcal{A},id}$. Similarly to the case of relational databases, the basic query class that we consider is the class of unions of conjunctive queries, which is a subclass of the class of First-Order Logic queries.

A First-Order Logic (FOL) *query* q over a *DL-Lite*$_{\mathcal{A},id}$ ontology $\mathcal{O}$ (resp., TBox $\mathcal{T}$) is a, possibly open, FOL formula $\varphi(\vec{x})$ whose predicate symbols are atomic concepts, value-domains, roles, or attributes of $\mathcal{O}$ (resp., $\mathcal{T}$). The free variables of $\varphi(\vec{x})$ are those appearing in $\vec{x}$, which is a tuple of (pairwise distinct) variables. In other words, the atoms of $\varphi(\vec{x})$ have the form $A(x)$, $D(x)$, $P(x, y)$, $U(x, y)$, or $x = y$, where:

- A, F, P, and U are respectively an atomic concept, a value-domain, an atomic role, and an atomic attribute in $\mathcal{O}$,
- x, y are either variables in $\vec{x}$ or constants in Γ.

$(it2009^{\mathcal{I}}) \in League^{\mathcal{I}}$
$(r7^{\mathcal{I}}) \in Round^{\mathcal{I}}$
$(r8^{\mathcal{I}}) \in Round^{\mathcal{I}}$
$(m7RJ^{\mathcal{I}}) \in PlayedMatch^{\mathcal{I}}$
$(m8NT^{\mathcal{I}}) \in Match^{\mathcal{I}}$
$(m8RM^{\mathcal{I}}) \in Match^{\mathcal{I}}$
$(roma^{\mathcal{I}}) \in Team^{\mathcal{I}}$
$(r7^{\mathcal{I}}, \texttt{"7"}) \in \mathbf{code}^{\mathcal{I}}$
$(r8^{\mathcal{I}}, \texttt{"8"}) \in \mathbf{code}^{\mathcal{I}}$

$(r7^{\mathcal{I}}, it2009^{\mathcal{I}}) \in BELONGS\text{-}TO^{\mathcal{I}}$
$(r8^{\mathcal{I}}, it2009^{\mathcal{I}}) \in BELONGS\text{-}TO^{\mathcal{I}}$
$(m7RJ^{\mathcal{I}}, roma^{\mathcal{I}}) \in HOME^{\mathcal{I}}$
$(m8NT^{\mathcal{I}}, napoli^{\mathcal{I}}) \in HOME^{\mathcal{I}}$
$(m8RM^{\mathcal{I}}, roma^{\mathcal{I}}) \in HOME^{\mathcal{I}}$
$(napoli^{\mathcal{I}}) \in Team^{\mathcal{I}}$
$(m7RJ^{\mathcal{I}}, \texttt{"RJ"}) \in \mathbf{code}^{\mathcal{I}}$
$(m8NT^{\mathcal{I}}, \texttt{"NT"}) \in \mathbf{code}^{\mathcal{I}}$
$(m8RM^{\mathcal{I}}, \texttt{"RM"}) \in \mathbf{code}^{\mathcal{I}}$

$(m7RJ^{\mathcal{I}}, r7^{\mathcal{I}}) \in PLAYED\text{-}IN^{\mathcal{I}}$
$(m8NT^{\mathcal{I}}, r8^{\mathcal{I}}) \in PLAYED\text{-}IN^{\mathcal{I}}$
$(m8RM^{\mathcal{I}}, r8^{\mathcal{I}}) \in PLAYED\text{-}IN^{\mathcal{I}}$
$(m7RJ^{\mathcal{I}}, juventus^{\mathcal{I}}) \in HOST^{\mathcal{I}}$
$(m8NT^{\mathcal{I}}, torino^{\mathcal{I}}) \in HOST^{\mathcal{I}}$
$(m8RM^{\mathcal{I}}, milan^{\mathcal{I}}) \in HOST^{\mathcal{I}}$
$(juventus^{\mathcal{I}}) \in Team^{\mathcal{I}}$
$(m7RJ^{\mathcal{I}}, \texttt{5/4/09}) \in \mathbf{date}^{\mathcal{I}}$
$(m7RJ^{\mathcal{I}}, \texttt{3}) \in \mathbf{homeGoals}^{\mathcal{I}}$
$(m7RJ^{\mathcal{I}}, \texttt{1}) \in \mathbf{hostGoals}^{\mathcal{I}}$

Figure 13 – A model of the ABox $\mathcal{A}_{fbc}$ for the football championship example.

The *arity* of q is the arity of $\vec{x}$. A query of arity 0 is called a *boolean query*. When we want to make the arity of a query q explicit, we denote the query as $q(\vec{x})$.

A *conjunctive query* (CQ) $q(\vec{x})$ over a *DL-Lite*$_{\mathcal{A},id}$ ontology is a FOL query of the form

$$\exists \vec{y}. \, conj(\vec{x}, \vec{y}),$$

where $\vec{y}$ is a tuple of pairwise distinct variables not occurring among the free variables $\vec{x}$, and where $conj(\vec{x}, \vec{y})$ is a *conjunction* of atoms. The variables $\vec{x}$ are also called *distinguished* and the (existentially quantified) variables $\vec{y}$ are called *non-distinguished*.

A *union of conjunctive queries* (UCQ) is a FOL query that is the disjunction of a set of CQs of the same arity, i.e., it is a FOL formula of the form:

$$\exists \vec{y}_1. \, conj_1(\vec{x}, \vec{y}_1) \vee \cdots \vee \exists \vec{y}_n. \, conj_n(\vec{x}, \vec{y}_n).$$

Given an interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$, the FOL query $q = \varphi(\vec{x})$ is interpreted in $\mathcal{I}$ as the set $q^{\mathcal{I}}$ of tuples $\vec{o} \in \Delta^{\mathcal{I}} \times \cdots \times \Delta^{\mathcal{I}}$ such that the formula φ evaluates to true in $\mathcal{I}$ under the assignment that assigns each object in $\vec{o}$ to the corresponding variable in $\vec{x}$ [2]. We call $q^{\mathcal{I}}$ the *answer* to q over $\mathcal{I}$. Notice that the answer to a boolean query is either the empty tuple, "()", considered as *true*, or the empty set, considered as *false*.

We remark that a relational database (over the atomic concepts, roles, and attributes) corresponds to a finite interpretation. Hence the notion of answer to a query introduced here is the standard notion of answer to a query evaluated over a relational database.

Example 4.2 Consider again the ontology $\mathcal{O}_{fbc} = \langle \mathcal{T}_{fbc}, \mathcal{A}_{fbc} \rangle$ introduced in Example 4.1, and the following query asking for all matches:

$$q_1(x) \leftarrow Match(x).$$

If $\mathcal{I}$ is the interpretation shown in Figure 13, we have that:

$$q_1^{\mathcal{I}} = \{(m8NT^{\mathcal{I}}), (m8RM^{\mathcal{I}})\}.$$

Notice that $\mathcal{I}$ is a model of $\mathcal{A}_{fbc}$, but not of $\mathcal{T}_{fbc}$. Let instead $\mathcal{I}'$ be the interpretation analogous to $\mathcal{I}$, but extended in such a way that it becomes also a model of $\mathcal{T}_{fbc}$, and hence of $\mathcal{O}_{fbc}$, as shown in Example 4.1. Then we have that:

$$q_1^{\mathcal{I}'} = \{(m8NT^{\mathcal{I}}), \ (m8RM^{\mathcal{I}}), \ (m7RJ^{\mathcal{I}})\}.$$

Suppose now that we ask for teams, together with the code of the match in which they have played as home team:

$$q_2(t, c) \leftarrow Team(t), HOME(m, t), Match(m), \mathbf{code}(m, c).$$

Then we have that

$$\begin{array}{rcl} q_2^{\mathcal{I}} & = & \{(napoli^{\mathcal{I}}, \texttt{"NT"}), \ (roma^{\mathcal{I}}, \texttt{"RM"})\}, \\ q_2^{\mathcal{I}'} & = & \{(roma^{\mathcal{I}}, \texttt{"RJ"}), \ (napoli^{\mathcal{I}}, \texttt{"NT"}), \ (roma^{\mathcal{I}}, \texttt{"RM"})\}. \end{array}$$

■

Given a query over a *DL-Lite*$_{\mathcal{A},id}$ ontology, we are interested in those answers to the query that are obtained for *all* possible databases (including infinite ones) that are models of the ontology. This corresponds to the fact that the ontology conveys only incomplete information about the domain of interest, and we want to guarantee that the answers to a query that we obtain are *certain*, independently of how we complete this incomplete information. This leads us to the following definition of *certain answers* to a query over an ontology.

Definition 4.2 *Let $\mathcal{O}$ be a DL-Lite$_{\mathcal{A},id}$ ontology and q a UCQ over $\mathcal{O}$. A tuple $\vec{c}$ of constants appearing in $\mathcal{O}$ is a* certain answer *to q over $\mathcal{O}$, written $\vec{c} \in cert(q, \mathcal{O})$, if for every model $\mathcal{I}$ of $\mathcal{O}$, we have that $\vec{c}^{\mathcal{I}} \in q^{\mathcal{I}}$.*

Answering a query q posed to an ontology $\mathcal{O}$ means exactly to compute the set of certain answers to q over $\mathcal{O}$.

Example 4.3 *Consider again the ontology introduced in Example 4.1, and queries q_1 and q_2 introduced in Example 4.2. One can easily verify that*

$$cert(q_1, \mathcal{O}) = \{(m8NT^{\mathcal{I}}),\ (m8RM^{\mathcal{I}}),\ (m7RJ^{\mathcal{I}})\},$$
$$cert(q_2, \mathcal{O}) = \{(roma^{\mathcal{I}}, \texttt{"RJ"}),\ (napoli^{\mathcal{I}}, \texttt{"NT"}),\ (roma^{\mathcal{I}}, \texttt{"RM"})\}.$$

■

Notice that, in the case where $\mathcal{O}$ is an unsatisfiable ontology, the set of certain answers to a (U)CQ q is the finite set of all possible tuples of constants whose arity is the one of q.

Reasoning Services. In studying *DL-Lite$_{\mathcal{A},id}$*, we are interested in several reasoning services, including the traditional DL reasoning services. Specifically, we consider the following problems for *DL-Lite$_{\mathcal{A},id}$* ontologies:

- *Ontology satisfiability*, i.e., given an ontology $\mathcal{O}$, verify whether $\mathcal{O}$ admits at least one model.
- *Concept and role satisfiability*, i.e., given a TBox $\mathcal{T}$ and a concept C (resp., a role R), verify whether $\mathcal{T}$ admits a model $\mathcal{I}$ such that $C^{\mathcal{I}} \neq \emptyset$ (resp., $R^{\mathcal{I}} \neq \emptyset$).
- We say that an ontology $\mathcal{O}$ (resp., a TBox $\mathcal{T}$) *logically implies* an assertion α, denoted $\mathcal{O} \models \alpha$ (resp., $\mathcal{T} \models \alpha$, if every model of $\mathcal{O}$ (resp., $\mathcal{T}$) satisfies α. The problem of *logical implication of assertions* consists of the following sub-problems:
  - *instance checking*, i.e., given an ontology $\mathcal{O}$, a concept C and a constant a (resp., a role R and a pair of constants a_1 and a_2), verify whether $\mathcal{O} \models C(a)$ (resp., $\mathcal{O} \models R(a_1, a_2)$);
  - *subsumption of concepts or roles*, i.e., given a TBox $\mathcal{T}$ and two general concepts C_1 and C_2 (resp., two general roles R_1 and R_2), verify whether $\mathcal{T} \models C_1 \sqsubseteq C_2$ (resp., $\mathcal{T} \models R_1 \sqsubseteq R_2$);
  - *checking functionality*, i.e., given a TBox $\mathcal{T}$ and a basic role Q, verify whether $\mathcal{T} \models (\mathsf{funct}\ Q)$.
  - *checking an identification constratins*, i.e., given a TBox $\mathcal{T}$ and an identification constraint $(\mathsf{id}\ C\ \pi_1, \ldots, \pi_n)$, verify whether $\mathcal{T} \models (\mathsf{id}\ C\ \pi_1, \ldots, \pi_n)$.

In addition we are interested in:

- *Query answering*, i.e., given an ontology $\mathcal{O}$ and a query q (either a CQ or a UCQ) over $\mathcal{O}$, compute the set $cert(q, \mathcal{O})$.

The notion of FOL-rewritability. We now introduce the notion of FOL-rewritability for both satisfiability and query answering, which will be used in the sequel.

First, given an ABox $\mathcal{A}$ (of the kind considered above), we denote by $DB(\mathcal{A}) = \langle \Delta^{DB(\mathcal{A})}, \cdot^{DB(\mathcal{A})} \rangle$ *the interpretation* defined as follows:

- $\Delta^{DB(\mathcal{A})}$ is the non-empty set consisting of the union of the set of all object constants occurring in $\mathcal{A}$ and the set $\{val(c) \mid c \text{ is a value constant that occurs in } \mathcal{A}\}$,
- $a^{DB(\mathcal{A})} = a$, for each object constant a,
- $A^{DB(\mathcal{A})} = \{a \mid A(a) \in \mathcal{A}\}$, for each atomic concept A,

- $P^{DB(\mathcal{A})} = \{(a_1, a_2) \mid P(a_1, a_2) \in \mathcal{A}\}$, for each atomic role P, and
- $U^{DB(\mathcal{A})} = \{(a, val(c)) \mid U(a, c) \in \mathcal{A}\}$, for each atomic attribute U.

Observe that the interpretation $DB(\mathcal{A})$ is a minimal model of the ABox $\mathcal{A}$.

Intuitively, FOL-rewritability of satisfiability (resp., query answering) captures the property that we can reduce satisfiability checking (resp., query answering) to evaluating a FOL query over the ABox $\mathcal{A}$ considered as a relational database, i.e., over $DB(\mathcal{A})$. The definitions follow.

Definition 4.3 *Satisfiability in a DL $\mathcal{L}$ is* FOL-rewritable*, if for every TBox $\mathcal{T}$ expressed in $\mathcal{L}$, there exists a boolean FOL query q, over the alphabet of $\mathcal{T}$, such that for every non-empty ABox $\mathcal{A}$, the ontology $\langle \mathcal{T}, \mathcal{A} \rangle$ is satisfiable if and only if q evaluates to false in $DB(\mathcal{A})$.*

Definition 4.4 *Answering UCQs in a DL $\mathcal{L}$ is* FOL-rewritable*, if for every UCQ q and every TBox $\mathcal{T}$ expressed over $\mathcal{L}$, there exists a FOL query q_1, over the alphabet of $\mathcal{T}$, such that for every non-empty ABox $\mathcal{A}$ and every tuple of constants $\vec{a}$ occurring in $\mathcal{A}$, we have that $\vec{a} \in cert(q, \langle \mathcal{T}, \mathcal{A} \rangle)$ if and only if $\vec{a}^{DB(\mathcal{A})} \in q_1^{DB(\mathcal{A})}$.*

We remark that FOL-rewritability of a reasoning problem that involves the ABox of an ontology (such as satisfiability or query answering) is tightly related to the data complexity of the problem. Indeed, since the FOL query considered in the above definitions depends only on the TBox (and the query), but not on the ABox, and since the evaluation of a First-Order Logic query (i.e., an SQL query without aggregation) over an ABox is in AC^0 in data complexity [2], FOL-rewritability of a problem has as an immediate consequence that the problem is in AC^0 in data complexity. Hence, one way of showing that for a certain DL $\mathcal{L}$ a problem is *not* FOL-rewritable, is to show that the data complexity of the problem for the DL $\mathcal{L}$ is above AC^0, e.g., LogSpace-hard, NLogSpace-hard, PTime-hard, or even coNP-hard. Several results on this problem can be found in [39, 7].

4.6 Mappings

In this section we discuss how to link external source databases with the elements of the ontology in OBDA. This problem is strictly connected to the problem of relating autonomous data sources to a global reconciled schema studied in the field of data integration [108, 87, 89], or the problem of moving data from a target to a source studied in data exchange [74, 76, 5]. To provide a complete picture of the various challenges that need to be addressed to effectively (and efficiently) solve this problem, we start with a brief overview of the classical forms of mappings that have been considered in the data integration setting.

4.6.1 Data Integration

Data integration is the problem of combining data residing at different sources, and providing the user with a unified view of these data [87, 99, 136, 108]. Such unified view represents the intensional level of the integrated and reconciled data, and provides the elements for expressing user queries. In formulating the queries, the user is freed from the knowledge on where data are, how data are structured at the sources, and how data are to be merged and reconciled to fit into the unified view.

Starting from the late 90s, research in data integration has mostly focused on declarative approaches (as opposed to procedural ones) [135, 108]. One of the outcomes of this research work is a clear architecture for (mediator-based[4]) data integration. According to this architecture [108], the main components of a data integration system are the *global schema*, the *sources*, and the *mapping*. Roughly speaking, the sources represent the repositories where the data are, the global schema represents the unified structure presented to the client, and the mapping relates the source data with the global schema.

[4]Other architectures, e.g. [16], are outside the scope of this report.

Modeling data integration systems. Sources that are to be integrated in a data integration system are typically heterogeneous, meaning that they adopt different models and systems for storing data. This poses challenging problems in both representing the sources in a common format within the integration system, and specifying the global schema. As for the former issue, data integration systems make use of suitable software components, called *wrappers*, that present data at the sources in the form adopted within the system, hiding the original structure of the sources and the way in which they are modelled. According to this consideration, we assume in what follows that the sources are always represented through a relational schema, and that non relational sources are translated in the relational model through specific wrappers.

For what concerns the specification of the global schema, there are at least two different approaches to the design of the global schema. In the first approach, the global schema is expressed in terms of a database model (e.g., the relational model, or a semistructured data model), and represents a sort of unified data structure accommodating the various data at the sources. In the second approach the global schema provides a conceptual representation of the application domain [28], rather than a specification of a data structure. Thus, in this approach the distinction between the global schema and the data sources reflects the separation between the conceptual level (the one presented to the client), and the logical/physical level of the information system (the one stored in the sources), with the mapping acting as the reconciling structure between the two levels. We notice that this second approach is the one we have already discussed in this report under the name of Ontology-base data access.

However, the design of the global schema is not the only issue in modelling a data integration system. Indeed, another crucial aspect is the definition of the *mapping*, i.e., the specification of the relationship holding between the sources and the global schema. To this purpose, two basic approaches have been proposed in the literature: the *local-as-view* (or simply LAV) approach, and the *global-as-view* (or simply GAV) approach.

The LAV approach requires the global schema to be specified independently of the sources. In turn, the sources are defined as views, i.e., queries, over the global schema [103, 1, 48]. In particular, to each element of the source schema, a view over the global schema is associated. Thus, in the local-as-view approach, we specify the meaning of the sources in terms of the elements of the global schema. The following example, adapted from [109], shows a LAV system in the relational setting.

Example 4.4 Consider a relational global schema with the following relations:

$$\begin{array}{l} \mathit{movie}(\mathit{Title}, \mathit{Year}, \mathit{Director}) \\ \mathit{european}(\mathit{Director}) \\ \mathit{review}(\mathit{Title}, \mathit{Critique}) \end{array}$$

The source schema consists of three source relations s_1, s_2, and s_3: s_1 stores title, year and director of movies produced since 1960, while s_2 stores title and critique of movies with European director, and s_3 stores European directors.

$$\begin{array}{l} s_1(\mathit{Title}, \mathit{Year}, \mathit{Director}) \\ s_2(\mathit{Title}, \mathit{Critique}) \\ s_3(\mathit{Director}) \end{array}$$

The LAV mapping is given by the following views (in this example we use conjunctive queries with arithmetic comparisons [2]):

$$\begin{array}{rcl} s_1(X, Y, Z) & \leftarrow & \mathit{movie}(X, Y, Z), (Y \geq 1960) \\ s_2(X, W) & \leftarrow & \mathit{movie}(X, Y, Z), \mathit{review}(X, W), \mathit{european}(Z) \\ s_3(X) & \leftarrow & \mathit{european}(X) \end{array}$$

Examples of LAV systems are Infomaster [68, 67], Information Manifold [103], and the system presented in [112].

The GAV approach requires that the global schema is expressed in terms of the data sources. More precisely, to every element of the global schema, a view over the data sources is associated, so that its meaning is specified in terms of the data residing at the sources, as shown in the following example.

Example 4.5 Consider a data integration system with source and global schemas as in Example 4.4. The GAV mapping associates to each relation symbol of the global schema a view over the sources as follows:

$$\begin{array}{rcl} \mathit{movie}(X,Y,Z) & \leftarrow & s_1(X,Y,Z) \\ \mathit{european}(X) & \leftarrow & s_3(X) \\ \mathit{review}(X,Y) & \leftarrow & s_2(X,Y) \end{array}$$

Examples of data integration systems based on GAV are TSIMMIS [78], Garlic [53], Squirrel [144], MOMIS [15], DIKE [120] and IBIS [34].

It should be easy to see that the LAV approach favors the extensibility of the integration system, and provides a more appropriate setting for its maintenance. For example, adding a new source to the system requires only to provide the definition of the source, and does not necessarily involve changes in the global schema. On the contrary, in the GAV approach, adding a new source typically requires changing the definition of the concepts in the global schema. On the other hand, defining the views associated to the global elements implies precisely understanding the relationships among the sources, that is in general a non-trivial task. A comparison between the two approaches can be found reported in [136, 109, 108, 32].

A generalization of both LAV and GAV mapping is the so-called *Global-Local-As-View* (GLAV) mapping [108, 77, 73, 75], where a query over the source schema is put in correspondence with a query over the global schema, i.e., a mapping assertion has the general form $q_{\S} \rightsquigarrow q_{\mathcal{G}}$. Depending on the form of $q_{\S}$ and $q_{\mathcal{G}}$, a GLAV mapping assertion can be specialized into a GAV or a LAV mapping assertion.

Querying Data Integration Systems. The run-time task that has been mainly studied in data integration is query processing. We briefly point out in the following some challenges arising when processing queries, and discuss various ways have been adopted in the literature to face this problem.

First of all consider the case in which the global schema is a flat relational database, i.e., no complex knowledge of the domain is represented through it. Query processing amounts therefore to computing the answer to queries posed in terms of the virtual global schema only on the basis of the data residing at the sources. The main issue is that the system should be able to re-express such queries in terms of a suitable set of queries posed to the sources, hand them to the sources, and assemble the results into the final answer.

It is worth noticing that, whereas query processing in the LAV approach has been always regarded as a difficult task, this problem has initially been considered much easier in the GAV approach, where it has been assumed that answering a query means *unfolding* its atoms according to their definitions on the sources. In fact, when the global schema is a flat relational database schema, query processing can be really reduced to unfolding. Instead, in LAV, the views in the mapping provide in general only a partial knowledge about the data that satisfy the global schema, hence query processing is inherently a form of reasoning in the presence of incomplete information [101, 1].

In other words, since several possibilities of populating the global schema with respect to the source extensions may exist, the semantics of a LAV system has to be given in terms of several database instances for the global schema, which have to be taken into account in processing a user query. We say that each such database satisfies the global schema with respect to the mapping. Conversely, in GAV, the mapping essentially specifies a single database for the global schema: evaluating the query over this database is equivalent to evaluating its unfolding over the sources [78, 15]. We will see later that this is in fact true only when global schema are flat, i.e., they do not provide integrity constraints.

Example 4.6 Consider again Example 4.5, and suppose we have the following user query:

$$q(X, Z) \leftarrow movie(X, 1998, Y), review(X, Z)$$

asking for title and critique of movies produced in 1998. The unfolding produces the query

$$q(X, Z) \leftarrow s_1(X, 1998, Y), s_2(X, Z)$$

The unfolded query can be directly evaluated on the sources, thus retrieving the answers.

In the LAV approach, query processing has been traditionally solved by means of *query rewriting*, where query processing is forced to be performed in two steps: in the first step the query is reformulated in terms of the views, and in the second the obtained query is evaluated on the view extensions, i.e., a database instance for the source schema. In query rewriting we want to reformulate the user query in terms of a *fixed language* referring to the alphabet used for specifying the source schema; the problem is that, since such language is fixed, and often coincides with the language used for the user query, there may be no rewriting that is *equivalent* to the original query. To face this problem, works on the LAV approach have originally concentrated on computing the *maximally contained rewriting*, that is the "best" possible rewriting, in the sense that it contains all other rewritings for the given query.

Example 4.7 Consider again Example 4.4, and suppose to have the same user query of Example 4.6:

$$q(X, Z) \leftarrow movie(X, 1998, Y), review(X, Z)$$

A rewriting of such a query is

$$q_r(X, Z) \leftarrow s_1(X, 1998, Y), s_2(X, Z)$$

Indeed, the body of q_r refers only to the source relations, and hence can be directly evaluated over the source extensions. Moreover, if we unfold q_r according to the view definitions we obtain the following query over the global schema

$$q_r(T, R) \leftarrow movie(T, 1998, D), movie(T, Y, D'), review(T, R), european(D')$$

in which we deleted the atom $1998 \geq 1960$, which evaluate to *true*. It is immediate to verify that q_r is contained in q. Furthermore q_r is the maximally contained rewriting.

A different approach, more general than query rewriting, consists in not posing any limitation on how the query is to be processed: all possible information, in particular the view extensions, can be used now for computing the answers to the query. This approach is commonly called *query answering*. We point out that the ultimate goal of query answering is to provide the *certain answers* to a user query, i.e., compute the intersection of the answer sets obtained by evaluating the query over any database that satisfies the global schema. Therefore, the notions of query answering and query rewriting are different, and we cannot always conclude that the evaluation on the sources of the maximally contained rewriting returns the certain answers to the user query. Nonetheless, this property holds for the common and widely studied case in which the views in the mapping are conjunctive queries [49]. Rewritings that are able to solve the query answering problem, i.e., such that their evaluation over the sources return the certain answers to the original query, are called *perfect rewritings*.

It is worth noticing that more complex GAV frameworks have been then considered in the literature, in which the semantics of the system is given in terms of a set of databases rather than the single database constructed according to the mapping, and techniques for query answering have been proposed that are more involved than simple unfolding [31, 33]. These approaches are tightly connected with the problem of query processing under integrity constraints in the presence of incomplete data.

Dealing with incomplete data. As already said, query processing in LAV has been traditionally considered a form of reasoning in the presence of incomplete information. Hence, sources in LAV data integration systems are generally assumed to be *sound*, but not necessarily complete, i.e., each source concept is assumed to store only a subset of the data that satisfy the corresponding view on the global schema. Not the same approach has been followed for processing queries in GAV, where, actually, the form of the mapping straightforwardly allows for the computation of a global database instance over which the user queries can be directly evaluated. Notice that proceeding in this way is analogous to unfolding the user queries as described in the above section.

Example 4.8 Consider again Example 4.5, and suppose to have the following source extension $\mathcal{D} = \{s_1(\mathsf{Platoon}, 1980, \mathsf{O.\ Stone}), s_3(\mathsf{F.\ Fellini})\}$, and a user query $q(Z) \leftarrow movie(X, Y, Z)$ asking for all the directors in the relation *movie*. It is easy, according to the GAV mapping, to construct the global database $\mathcal{B} = \{movie(\mathsf{Platoon}, 1986, \mathsf{O.\ Stone}), european(\mathsf{F.\ Fellini})\}$, over which we evaluate the user query, thus obtaining the answer set $\{\mathsf{O.\ Stone}\}$.

Answering a user query as done in the above example, actually means assuming that the views in the mapping are *exact*, i.e., that they provide exactly the data that satisfy the corresponding global relation. However, also in GAV systems it may happen that the sources provide only a subset of the data that satisfy the corresponding relations in the global schema, hence views in the mapping should be considered sound rather than exact. This becomes particularly relevant when integrity constraints are specified on the global schema, as shown in [33], and of course, when the language used to specify the global schema is a semantically rich language, such as a DL, as used in OBDA [124, 38, 37].

Example 4.9 Let us now continue Example 4.8 and assume that a constraint on the global schema schema imposes that every director in the relation *european* is a director of at least one movie. Assume also that the mapping is sound. In this case we know that Fellini is a director of at least one movie, even if we do not know which is the movie and in which year it has been realized. Nonetheless, under the sound assumption, the answer to our user query q asking for all directors should be now $\{\mathsf{O.\ Stone}, \mathsf{F.\ Fellini}\}$.

The above example shows that in the presence of incomplete data with respect to integrity constraints specified on the global schema, unfolding is in general not sufficient to answer a user query in GAV, whereas reasoning on the constraints is needed in order to compute the certain answers to the query. Moreover, it should be easy to see that reasoning on the constraints is needed also in the LAV approach: consider again Example 4.4, where now integrity constraints on the global schema, user query and source extensions are as in Example 4.8; without taking into account the inclusion dependency, we would not get the certain answers also in this case.

In conclusion, in order to model complex data integration scenarios, the specification of different forms of integrity constraints on the global schema should be allowed, and different assumptions on the mapping should be enabled, in both the LAV and the GAV framework. Roughly speaking, such assumptions indicate how to interpret the data that can be retrieved from the sources with respect to the data that satisfy the corresponding portion of the global schema, and allow for different forms of reasoning on the constraints.

Architecture for data integration. According to [108], we formalize a *data integration system* $\mathcal{J}$ in terms of a triple $\langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$, where

- $\mathcal{G}$ is the *global schema*, expressed in a language $\mathcal{L}_\mathcal{G}$ over an alphabet $\mathcal{A}_\mathcal{G}$. The alphabet comprises a symbol for each element of $\mathcal{G}$ (i.e., relation if $\mathcal{G}$ is relational, class if $\mathcal{G}$ is object-oriented, etc.).
- $\mathcal{S}$ is the *source schema*, expressed in a language $\mathcal{L}_\mathcal{S}$ over an alphabet $\mathcal{A}_\mathcal{S}$. The alphabet $\mathcal{A}_\mathcal{S}$ includes a symbol for each element of the sources.

- $\mathcal{M}$ is the *mapping* between $\mathcal{G}$ and $\mathcal{S}$, constituted by a set of *assertions* of the form $q_{\mathcal{S}} \rightsquigarrow q_{\mathcal{G}}$ where $q_{\mathcal{S}}$ and $q_{\mathcal{G}}$ are two queries of the same arity, respectively over the source schema $\mathcal{S}$, and over the global schema $\mathcal{G}$. Queries $q_{\mathcal{S}}$ are expressed in a query language $\mathcal{L}_{\mathcal{M},\mathcal{S}}$ over the alphabet $\mathcal{A}_{\mathcal{S}}$, and queries $q_{\mathcal{G}}$ are expressed in a query language $\mathcal{L}_{\mathcal{M},\mathcal{G}}$ over the alphabet $\mathcal{A}_{\mathcal{G}}$. Intuitively, an assertion $q_{\mathcal{S}} \rightsquigarrow q_{\mathcal{G}}$ specifies that the concept represented by the query $q_{\mathcal{S}}$ over the sources corresponds to the concept in the global schema represented by the query $q_{\mathcal{G}}$.

The semantics of a data integration system is based on the notion of interpretation in logic. Indeed, in this paper we assume that $\mathcal{G}$ is formalized as a logical theory, and therefore, given a source database D (i.e., a database for $\mathcal{S}$), the semantics of the whole system coincides with the set of interpretations that satisfy all the assertions of $\mathcal{G}$ (i.e., they are logical models of $\mathcal{G}$) and all the assertions of $\mathcal{M}$ with respect to D. The notion of mapping satisfaction depends on semantic assumptions adopted on the mapping, and on the languages used for specifying queries in the mapping. A common situation is that in which the queries in the mapping are expressed in (fragments of) first-order logic, and mappings are considered *sound*, i.e., an interpretation $\mathcal{I}$ satisfies a mapping $q_{\mathcal{S}} \rightsquigarrow q_{\mathcal{G}}$ with respect to a source database D if $q_{\mathcal{S}}^{D} \subseteq q_{\mathcal{G}}^{\mathcal{I}}$. Such a set of interpretations, denoted $sem_D(\mathcal{J})$, is called the set of models of $\mathcal{J}$ relative to D.

There are two basic tasks concerning a data integration system that we consider in this paper. The first task is relevant in the design phase of the system, and concerns the possibility of reasoning over the global schema: given $\mathcal{G}$ and a logical assertion α, check whether α holds in every model of $\mathcal{G}$. The second task is query answering, which is crucial during the run-time of the system. Queries to $\mathcal{J}$ are posed in terms of the global schema $\mathcal{G}$, and are expressed in a query language $\mathcal{L}_{\mathcal{Q}}$ over the alphabet $\mathcal{A}_{\mathcal{G}}$. A query is intended to provide the specification of which extensional information to extract from the domain of interest in the data integration system. More precisely, given a source database D, the answer $q^{\mathcal{J},D}$ to a query q in $\mathcal{J}$ with respect to D is the set of tuples t of objects such that $t \in q^{\mathcal{B}}$ (i.e., t is an answer to q over $\mathcal{B}$) for *every* model $\mathcal{B}$ of $\mathcal{J}$ relative to D. The set $q^{\mathcal{J},D}$ is called the set of *certain answers* to q in $\mathcal{J}$ with respect to D. Note that, from the point of view of logic, finding certain answers is a logical implication problem: check whether the fact that t satisfies the query logically follows from the information on the sources and on the mapping.

The above definition of data integration system is general enough to capture virtually all approaches in the literature. Obviously, the nature of a specific approach depends on the characteristics of the mapping, and on the expressive power of the various schema and query languages. For example, the language $\mathcal{L}_{\mathcal{G}}$ may be very simple (basically allowing for the definition of a set of relations), or may allow for various forms of integrity constraints to be expressed over the symbols of $\mathcal{A}_{\mathcal{G}}$. Analogously, the type (e.g., relational, semistructured, etc.) and the expressive power of $\mathcal{L}_{\mathcal{S}}$ varies from one approach to another.

4.6.2 Accessing Data through *DL-Lite*$_{\mathcal{A},id}$ Ontologies

We now illustrate a specific proposal of data integration system based on *DL-Lite*$_{\mathcal{A},id}$, by describing the three components of the system. The choice for the languages used in the three components is tailored towards the goal of an optimal trade-off between expressive power and complexity. After the description of the three components, we briefly illustrate the algorithm for answering queries in our approach, and we discuss its computational complexity.

The global schema. As said before, we follow an approach in which the global schema represents the conceptual model of the domain of interest, rather than a mere description of a unified view of the source data, as often done in the practice. We have also discussed the advantages of expressing the conceptual model in terms of a DL. Finally, we have seen that *DL-Lite*$_{\mathcal{A},id}$ represents a valuable choice as a formalism for expressing the global schema of a data integration system. Therefore, in our approach to data integration, the global schema is expressed in terms of a *DL-Lite*$_{\mathcal{A},id}$ TBox.

The source schema. If $\mathcal{J} = \langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$ is a data integration system in our approach, $\mathcal{S}$ is assumed to be a flat relational database schema, representing the schemas of all the data sources. Actually, this is not a limitation of the system, since the source schema can always be thought of as the schema produced through a wrapping the original data sources. This can be profitably realized by making use of existing data federation tools (IBM InfoSphere Federation Server[5], is an example of a commercial product for data federation, whereas Teiid[6] is an open source solution for federation, also called data virtualization).Specifically, we assume that a data federation tool is in charge of interacting with the data sources, presenting them as a single relational database schema. Such a schema is obtained by wrapping physical sources, possibly heterogeneous, and not necessarily in relational format. Furthermore, the data federation tool is in charge of answering queries formulated over the source schema, by suitably transforming such queries, forwarding them to the right sources, and finally combining the single results into the overall answer. In other words, the data federation tool makes the whole system independent from the physical nature of the sources, by providing a logical representation of them (physical independence), whereas the other components of the system make all the logical aspects transparent to the user, by maintaining the conceptual global schema separate from the logical federated schema, and connecting them via suitable mappings (logical independence).

The mapping. The mappings in our approach establish the relationship between the source schema and the global schema, thus specifying how data stored at the sources are linked to the instances of the concepts and the roles in the global schema. More specifically, we follow the *GAV (global-as-view)* approach for specifying mappings, which requires to describe the meaning of every element of the global schema by associating to it a view over the sources. The dual approach, called LAV (local-as-view), would require the sources to be defined as views over the global schema.

Moreover, our mapping specification language takes suitably into account the *impedance mismatch* problem, i.e., the mismatch between the way in which data is (and can be) represented in a data source, and the way in which the corresponding information is rendered through the global schema.

The mapping assertions keep data value constants separate from object identifiers, and construct identifiers as (logic) terms over data values. More precisely, object identifiers in our approach are *terms* of the form $f(d_1, \ldots, d_n)$, called *object terms*, where f is a function symbol of arity $n > 0$, and $d_1, \ldots, d_n$ are data values stored at the sources. Note that this idea traces back to the work done in deductive object-oriented databases [98].

We detail below the above ideas. The mapping component is specified through a set of mapping assertions, each of the form

$$\Phi(\vec{v}) \rightsquigarrow G(\vec{w}) \tag{2}$$

where

- $\Phi(\vec{v})$, called the *body* of the mapping, is a first-order logic (FOL) query of arity $n > 0$, with distinguished variables $\vec{v}$, over the source schema $\mathcal{S}$ (we will write such query in the SQL syntax), and
- $G(\vec{w})$, called the *head*, is an atom where G can be an atomic concept, an atomic role, or an atomic attribute occurring in the global schema $\mathcal{G}$, and $\vec{w}$ is a sequence of terms.

We define three different types of mapping assertions:

- *Concept mapping assertions*, in which the head is a unary atom of the form $A(f(\vec{v}))$, where A is an atomic concept and f is a function symbol of arity n;
- *Role mapping assertions*, in which the head is a binary atom of the form $P(f_1(\vec{v}'), f_2(\vec{v}''))$, where P is an atomic role, f_1 and f_2 are function symbols of arity $n_1, n_2 > 0$, and $\vec{v}'$ and $\vec{v}''$ are sequences of variables appearing in $\vec{v}$;

[5] `www.ibm.com/software/products/it/it/ibminfofedeserv/`
[6] `www.jboss.org/teiid`

- *Attribute mapping assertions*, in which the head is a binary atom of the form $U(f(\vec{v}'), v'')$, where U is an atomic attribute, f is a function symbol of arity $n' > 0$, $\vec{v}'$ is a sequence of variables appearing in $\vec{v}$, v'' is a variable appearing in $\vec{v}$.

In words, such mapping assertions are used to map source relations (and the tuples they store), to concepts, roles, and attributes of the ontology (and the objects and the values that constitute their instances), respectively. Note that an attribute mapping also specifies the type of values retrieved from the source database, in order to guarantee coherency of the system.

M_1: `SELECT T.mcode, T.round, T.league FROM TABLE T WHERE T.league > 0`	$\rightsquigarrow$ *match*(**m**(`T.mcode, T.round, T.league`))
M_2: `SELECT T.mcode, T.round, T.league FROM TABLE T WHERE T.league > 0`	$\rightsquigarrow$ **code**(**m**(`T.mcode, T.round, T.league`), `T.mcode`)
M_3: `SELECT T.mcode, T.round, T.league FROM TABLE T WHERE T.league > 0`	$\rightsquigarrow$ *PLAYED-IN*(**m**(`T.mcode, T.round, T.league`), **r**(`T.round, T.league`))

Figure 14 – Example of mapping assertions

Example 4.10 We refer again to the football domain. Suppose that the source schema contains the relational table `TABLE(mcode,league,round,home,host)`, where a tuple (m, l, r, h_1, h_2) with $l > 0$ represents a match with code m of league l and round r, and with home team h_1 and host team h_2. If we want to map the tuples from the table `TABLE` to the global schema shown in Figure 10, the mapping assertions might be as shown in Figure 14. M_1 is a concept mapping assertion that selects from `TABLE` the code and the round of matches (only for the appropriate tuples), and then uses such data to build instances of the concept *match*, using the function symbol **m**. M_2 is an attribute mapping assertion that is used to "populate" the attribute **code** for the objects that are instances of *match*. Finally, M_3 is a role mapping assertion relating `TABLE` to the atomic role *PLAYED-IN*, where instances of *round* are denoted by means of the function symbol **r**. We notice that in the mapping assertion M_2, the mapping designer had to specify a correct *DL-Lite*$_{\mathcal{A},id}$ data type for the values extracted from the source.

We point out that, during query answering, the body of each mapping assertion is never really evaluated in order to extract values from the sources to build instances of the global schema, but rather it is used to unfold queries posed over the global schema, rewriting them into queries posed over the source schema.

Semantics of *dlliteaid* data integration systems. As we already said when presenting the architecture for data integration, in order to define the semantics of a *DL-Lite*$_{\mathcal{A},id}$ data integration system $\langle \mathcal{T}, \mathcal{S}, \mathcal{M} \rangle$, we need to define when an interpretation *satisfies an assertion in* $\mathcal{M}$ *w.r.t. a database instance* D *for* $\mathcal{S}$. To this end, we make use of the notion of ground instance of a formula. Let $\Psi(\vec{x})$ be a formula over a *DL-Lite*$_{\mathcal{A},id}$ TBox with n distinguished variables $\vec{x}$, and let $\vec{t}$ be a tuple of value constants of arity n. Then the ground instance $\Psi[\vec{x}/\vec{t}]$ of $\Psi(\vec{x})$ is the formula obtained from $\Psi(\vec{x})$ by substituting every occurrence of x_i with v_i, for $i \in \{1, \ldots, n\}$. We are now ready to define when an interpretation satisfies a mapping assertion.

In the following, we denote with $ans(\varphi, D)$ the set of tuples (of the arity of φ) of value constants returned as the result of the evaluation of the SQL query φ over the database DB.

Definition 4.5 *Let $\mathcal{J} = \langle \mathcal{T}, \mathcal{S}, \mathcal{M} \rangle$ be a data integration system where $\mathcal{T}$ is a DL-Lite$_{\mathcal{A},id}$ ontology and $\mathcal{M}$ is a set of mapping assertions as defined above, let D be a database instance for $\mathcal{S}$, and let $\mathcal{I}$ be an interpretation of $\mathcal{J}$. Then, let m be an assertion in $\mathcal{M}$ of the form $\Phi(\vec{v}) \rightsquigarrow G(\vec{w})$, where $\vec{v}$, $\vec{w}$ are as in (2). We say that $\mathcal{I}$* satisfies *m* w.r.t. *D, if for every tuple of values $\vec{t}$ such that $\vec{t} \in ans(\Phi, D)$, and for every ground atom X in $\Psi[\vec{v}/\vec{t}]$, we have that:*

- *if X has the form $A(s)$, then $s^{\mathcal{I}} \in A^{\mathcal{I}}$;*
- *if X has the form $P(s_1, s_2)$, then $(s_1^{\mathcal{I}}, s_2^{\mathcal{I}}) \in P^{\mathcal{I}}$;*
- *if X has the form $U(s_1, s_2)$, then $(s_1^{\mathcal{I}}, s_2^{\mathcal{I}}) \in U^{\mathcal{I}}$.*

We say that $\mathcal{I}$ satisfies *$\mathcal{M}$* w.r.t. *D, if it satisfies every assertion in $\mathcal{M}$ w.r.t. D. We say that $\mathcal{I}$ is a* model *of $\mathcal{J}$ with respect to D if $\mathcal{I}$ is a model of $\mathcal{T}$ and satisfies $\mathcal{M}$ w.r.t. D. Finally, we denote with $Mod(\mathcal{J}, D)$ the set of models of $\mathcal{J}$ with respect to D, and we say that $\mathcal{J}$ is* satisfiable *with respect to D if $Mod(\mathcal{J}, D) \neq \emptyset$.*

Let q denote a UCQ expressed over the TBox $\mathcal{T}$ of $\mathcal{J}$. We call *certain answers to q over $\mathcal{J}$ with respect to a source database D*, denoted $certq\mathcal{J}D$, the set of n-tuples of terms in the set of constants Γ, defined as

$$certq\mathcal{J}D = \{\vec{t} \mid \vec{t}^{\mathcal{I}} \in q^{\mathcal{I}}, \text{ for all } \mathcal{I} \in Mod(\mathcal{J}, D)\}.$$

Given a query q over the TBox of an ontology-based data integration system and a source database D for J, *query answering* is the problem of computing the certain answers to q with respect to D.

4.7 Temporal Logics and Actions

To realize the KAB action component we need to express dynamics of the domain of interest in terms of full-fledged temporal logics such as LTL [123], CTL [14, 71], CTL* [55], μ-calculus [105, 72] for expressing sophisticated dynamic properties. However while such logics are normally considered for verification in their propositional variant, here we need to consider their First-Order variants so as to deal with data. Coping with both data and dynamics is very challenging in general and is one of the most important contributions that WP2 in ACSI is giving to the scientific community.

Here we concentrate on specification in temporal logics. Specification of semantic actions can be formalized following the literature on Reasoning about Actions in Artificial Intelligence, e.g., in the Situation Calculus [125], or high-level agent programming languages based on it, such as ConGolog [61].

Dynamic/temporal properties are often formulas expressed in (fragments of) the *computation tree logic* CTL*, a language that, interpreted on Kripke structures, allows for capturing temporal possibility and necessity of properties to happen over time, as the system evolves.

Despite their relative simplicity, CTL* and its most widely used fragments, the *branching-time* temporal logic CTL and the *linear-time* temporal logic LTL, proved successful in capturing many specifications of interest in the practice, as witnessed by their affirmation in industrial contexts for verification of both hardware and software (real) system components (see, e.g., [57] for a list of notable examples). More general specifications of transition systems can be expressed in propositional μ-calculus, a modal logic with least and greatest fixpoint operators, powerful enough to capture the whole CTL* [58]. Typically, the model of a system (i.e., its transition system) is not provided explicitly, i.e., by listing all of its nodes, transitions and labels, but in a compact way, using a suitable language that, when interpreted, *generates* the transition system. As mentioned, here we need to consider First-Order variants of these logics. We do so in the following.

4.7.1 Semantics via Transition Systems and Traces

The semantics of a process running over the Artifact Layer of an artifact-based system can be characterized by means of a possibly infinite transition system whose states are system's snapshots, i.e., collections of database instances related to artifact instances, event instances, environment gateways and relationships between them (see Section 2.2). In fact, according to Section 2.4, in the A^3M we assume that the evolution of the system determined by the process can be represented by means of a nondeterministic transition relation $\mathcal{F}$, which in turn induces a transition system representing all possible computations that the process can perform on the Artifact Layer of the system.

Technically, in our setting a transition system Υ has the form $\langle \Delta, \mathcal{R}, \Sigma, s_0, \mathit{snap}, \Longrightarrow \rangle$, where:

- Δ is a countably infinite set of values;
- $\mathcal{R}$ is a database schema;
- Σ is a set of states;
- $s_0 \in \Sigma$ is the initial state;
- *snap* is a function that, given a state $s \in \Sigma$, returns the snapshot associated to s, which is a database made up of values in Δ that conforms to schema $\mathcal{R}$;
- $\Longrightarrow \subseteq \Sigma \times \Sigma$ is a transition relation between pairs of states.

A *run* (or *trace*) τ in Υ is a (finite or infinite) sequence of states $s_0 s_1 s_2 \cdots$ rooted at s_0, where $s_i \Longrightarrow s_{i+1}$.

In the context of an A^3M artifact system, $\mathcal{R}$ must be intended as a global schema comprising the schemas of artifact types, event types, environment gateways as well as of the relationship stereotypes between them.

As discussed in Section 3.3, every snapshot at the Artifact Layer determines one at the Semantic Layer. This is obtained by applying the mappings that relate the database schemas used at the Artifact Layer to the knowledge component of the Semantic Layer's KAB. Consequently, the transition system Υ determines a corresponding transition system Υ_S at the Semantic Layer. Υ_S is structurally identical to Υ but differs in the *snap* function, which now extracts from a state its associated ABox over the KAB's knowledge component. The ABox associated to state s in Υ_S is obtained by applying the mappings on the snapshot extracted from the corresponding state s in Υ.

4.7.2 First-Order Variants of μ-Calculus

As already mentioned, a possibility for specifying rich dynamic properties over an artifact-based system is the μ-calculus [70, 133, 27], one of the most powerful temporal logics for which model checking has been investigated in the finite-state setting. Indeed, such a logic is able to express both linear time logics such as LTL and PSL, and branching time logics such as CTL and CTL* [56]. The main characteristic of μ-calculus is the ability of expressing directly least and greatest fixpoints of (predicate-transformer) operators formed using formulae relating the current state to the next one. By using such fixpoint constructs one can easily express sophisticated properties defined by induction or co-induction. This is the reason why virtually all logics used in verification can be considered as fragments of μ-calculus. From a technical viewpoint, μ-calculus separates local properties, i.e., properties asserted on the current state or on states that are immediate successors of the current one, and properties that talk about states that are arbitrarily far away from the current one [27]. The latter are expressed through the use of fixpoints.

The semantics of μ-calculus is defined over transition systems. Since in our context the states of such transition systems are associated to database instances, we are interested in first-order variants of the μ-calculus, where temporal modalities and fixpoints are intertwined with queries

$$
\begin{aligned}
(Q)^{\Upsilon}_{v,V} &= \{s \in \Sigma \mid \text{ANS}(Qv, snap(s))\} \\
(\neg\Phi)^{\Upsilon}_{v,V} &= \Sigma - (\Phi)^{\Upsilon}_{v,V} \\
(\Phi_1 \wedge \Phi_2)^{\Upsilon}_{v,V} &= (\Phi_1)^{\Upsilon}_{v,V} \cap (\Phi_2)^{\Upsilon}_{v,V} \\
(\exists x.\Phi)^{\Upsilon}_{v,V} &= \{s \in \Sigma \mid \exists t.t \in \Delta \text{ and } s \in (\Phi)^{\Upsilon}_{v[x/t],V}\} \\
(\langle - \rangle \Phi)^{\Upsilon}_{v,V} &= \{s \in \Sigma \mid \exists s'.s \Rightarrow s' \text{ and } s' \in (\Phi)^{\Upsilon}_{v,V}\} \\
(Z)^{\Upsilon}_{v,V} &= V(Z) \\
(\mu Z.\Phi)^{\Upsilon}_{v,V} &= \bigcap \{\mathcal{S} \subseteq \Sigma \mid (\Phi)^{\Upsilon}_{v,V[Z/\mathcal{S}]} \subseteq \mathcal{S}\}
\end{aligned}
$$

Figure 15 – Semantics of $\mu\mathcal{L}$.

over such database instances. Of particular interest is the support of first-order quantification across states, which is needed, for example, to express liveness properties that refer to the same data at various points in time (e.g. "if a customer order x is processed now, then x will eventually be shipped"). More specifically, we consider the following first-order variant of the μ-calculus [121], called $\mu\mathcal{L}$ and defined as follows:

$$\Phi ::= Q \mid \neg\Phi \mid \Phi_1 \wedge \Phi_2 \mid \exists x.\Phi \mid \langle - \rangle \Phi \mid Z \mid \mu Z.\Phi$$

where Q is a possibly open FO query, and Z is a second order predicate variable (of arity 0). We make use of the following abbreviations: $\forall x.\Phi = \neg(\exists x.\neg\Phi)$, $\Phi_1 \vee \Phi_2 = \neg(\neg\Phi_1 \wedge \neg\Phi_2)$, $[-]\Phi = \neg\langle - \rangle\neg\Phi$, and $\nu Z.\Phi = \neg\mu Z.\neg\Phi[Z/\neg Z]$.

As usual in μ-calculus, formulae of the form $\mu Z.\Phi$ (and $\nu Z.\Phi$) must obey to the *syntactic monotonicity* of Φ wrt Z, which states that every occurrence of the variable Z in Φ must be within the scope of an even number of negation symbols. This ensures that the least fixpoint $\mu Z.\Phi$ (as well as the greatest fixpoint $\nu Z.\Phi$) always exists.

Since $\mu\mathcal{L}$ also contains formulae with both individual and predicate free variables, given a transition system Υ, we introduce an individual variable valuation v, i.e., a mapping from individual variables x to Δ, and a predicate variable valuation V, i.e., a mapping from predicate variables Z to subsets of Σ. With these three notions in place, we assign meaning to formulae by associating to Υ, v, and V an *extension function* $(\cdot)^{\Upsilon}_{v,V}$, which maps formulae to subsets of Σ. Formally, the extension function $(\cdot)^{\Upsilon}_{v,V}$ is defined inductively as shown in Figure 15. In the Figure, as well as in the remainder of this Section, we use $t\theta$ (resp., $\varphi\theta$) to denote the term (resp., the formula) obtained by applying the substitution θ to t (resp., φ). Furthermore, given a first-order query Q and a database instance $\mathcal{I}$, the *answer* $\text{ANS}(Q, \mathcal{I})$ to Q over $\mathcal{I}$ is the set of assignments θ from the free variables of Q to the domain of $\mathcal{I}$, such that $\mathcal{I} \models Q\theta$. We treat $Q\theta$ as a boolean query, and with some abuse of notation, we say $\text{ANS}(Q\theta, \mathcal{I}) \equiv true$ if and only if $\mathcal{I} \models Q\theta$.

Example 4.11 *An example of $\mu\mathcal{L}$ formula is:*

$$\exists x_1, \ldots, x_n. \bigwedge_{i \neq j} x_i \neq x_j \wedge \bigwedge_{i \in \{1,\ldots,n\}} \mu Z.[Stud(x_i) \vee \langle - \rangle Z]$$

The formula asserts that there are at least n distinct objects/values, each of which eventually denotes a student along some execution path. Notice that the formula does not imply that all of these students will be in the same state, nor that they will all occur in a single run. It only says that in the entire transition systems there are (at least) n distinct students.

When Φ is a closed formula, $(\Phi)^{\Upsilon}_{v,V}$ depends neither on v nor on V, and we denote the extension of Φ simply by $(\Phi)^{\Upsilon}$. We say that a closed formula Φ holds in a state $s \in \Sigma$ if $s \in (\Phi)^{\Upsilon}$. In this case, we write $\Upsilon, s \models \Phi$. We say that a closed formula Φ holds in Υ, denoted

by $\Upsilon \models \Phi$, if $\Upsilon, s_0 \models \Phi$, where s_0 is the initial state of Υ. We call *model checking* verifying whether $\Upsilon \models \Phi$ holds. In particular we are interested in formally verifying dynamic properties of an artifact-based system. Given the transition system Υ of an artifact-based system and a dynamic property Φ expressed in $\mu\mathcal{L}$, we say that Υ *verifies* Φ if

$$\Upsilon \models \Phi.$$

The challenging point is that Υ is in general-infinite state, so in principle one would like to devise a finite-state transition system which is a faithful abstraction of Υ, in the sense that it preserves the truth value of all $\mu\mathcal{L}$ formulae. Unfortunately, this program is doomed right from the start if we insist on using full $\mu\mathcal{L}$ as the verification formalism. Indeed formulae of the kind shown in Example (4.11) defeat any kind of finite-state transition system. To overcome this issue, fragments of $\mu\mathcal{L}$ can be investigated, mediating between the required expressivity and the possibility of finding suitable finite-state abstractions of the transition system at hand. In the following, we present two interesting sublogics of $\mu\mathcal{L}$ that go in this direction.

4.7.3 History Preserving Mu-Calculus

The first fragment of $\mu\mathcal{L}$ that we consider is $\mu\mathcal{L}_A$, which is characterized by the assumption that quantification over individuals is restricted to individuals that are present in the current database. To enforce such a restriction, we introduce a special predicate $\textsc{live}(x)$, which states that x belongs to the current active domain. The logic $\mu\mathcal{L}_A$ is defined as follows:

$$\Phi ::= Q \mid \neg\Phi \mid \Phi_1 \wedge \Phi_2 \mid \exists x.\textsc{live}(x) \wedge \Phi \mid \langle - \rangle \Phi \mid Z \mid \mu Z.\Phi$$

We make use of the usual abbreviation, including $\forall x.\textsc{live}(x) \rightarrow \Phi = \neg(\exists x.\textsc{live}(x) \wedge \neg\Phi)$. Formally, the extension function $(\cdot)^{\Upsilon}_{v,V}$ is defined inductively as in Figure 15, with the new special predicate $\textsc{live}(x)$ interpreted as follows:

$$(\textsc{live}(x))^{\Upsilon}_{v,V} = \{s \in \Sigma \mid x/d \in v \text{ implies } d \in \textsc{adom}(\mathit{snap}(s))\}$$

Example 4.12 *As an example, consider the following $\mu\mathcal{L}_A$ formula:*

$$\begin{aligned} \nu X.(\forall x.\textsc{live}(x) \wedge \mathit{Stud}(x) \rightarrow \\ \mu Y.(\exists y.\textsc{live}(y) \wedge \mathit{Grad}(x,y) \vee \langle - \rangle Y) \wedge [-]X), \end{aligned}$$

which states that, along every path, it is always true, for each student x, that there exists an evolution that eventually leads to a graduation of the student (with some final mark y).

4.7.4 Persistence Preserving Mu-Calculus

The second fragment of $\mu\mathcal{L}$ that we consider is $\mu\mathcal{L}_P$, which further restricts $\mu\mathcal{L}_A$ by requiring that individuals over which we quantify must continuously persists along the system evolution for the quantification to take effect.

With a slight abuse of notation, in the following we write $\textsc{live}(x_1, \ldots, x_n) = \bigwedge_{i \in \{1,\ldots,n\}} \textsc{live}(x_i)$.

The logic $\mu\mathcal{L}_P$ is defined as follows:

$$\begin{aligned} \Phi ::= Q \mid \neg\Phi \mid \Phi_1 \wedge \Phi_2 \mid \exists x.\textsc{live}(x) \wedge \Phi \mid \langle - \rangle(\textsc{live}(\vec{x}) \wedge \Phi) \mid \\ [-](\textsc{live}(\vec{x}) \wedge \Phi) \mid Z \mid \mu Z.\Phi \end{aligned}$$

where Q is a possibly open FO query, Z is a second order predicate variable, and the following assumption holds: in $\textsc{live}(\vec{x}) \wedge \langle - \rangle\Phi$ and $\textsc{live}(\vec{x}) \wedge [-]\Phi$, the variables $\vec{x}$ are exactly the free variables of Φ, with the proviso that we substitute to each bounded predicate variable Z in Φ its bounding formula $\mu Z.\Phi'$. We use the usual abbreviations, including: $\langle - \rangle(\textsc{live}(\vec{x}) \rightarrow \Phi) = \neg[-](\textsc{live}(\vec{x}) \wedge \neg\Phi)$ and $[-](\textsc{live}(\vec{x}) \rightarrow \Phi) = \neg\langle - \rangle(\textsc{live}(\vec{x}) \wedge \neg\Phi)$. Intuitively, the use of $\textsc{live}(\cdot)$

in $\mu\mathcal{L}_P$ ensures that individuals are only considered if they persist along the system evolution, while the evaluation of a formula with individuals that are not present in the current database trivially leads to false or true (depending on the use of negation).

This fragment is of particular interest in situations where the domain values are used as references to external objects/individuals belonging to the system. In such a situation, it could be the case that a value maintains the identity of the reference over time only until it persists in the system, but once it disappears, it could later on be used to refer to another, distinct object/individual. This is the case, for example, of artifact instance identifiers: once an artifact instance its destroyed and all references to it are removed, the same identifier could later on be used to denote a different artifact instance.

Example 4.13 *Getting back to the example above, its variant in $\mu\mathcal{L}_P$ is*

$$\nu X.(\forall x.\textsc{live}(x) \wedge Stud(x) \rightarrow \\ \mu Y.(\exists y.\textsc{live}(y) \wedge Grad(x,y) \vee \langle - \rangle(\textsc{live}(x) \wedge Y)) \wedge [-]X)$$

which states that, along every path, it is always true, for each student x, that there exists an evolution in which x persists in the database until she eventually graduates (with some final mark y). Formula

$$\nu X.(\forall x.\textsc{live}(x) \wedge Stud(x) \rightarrow \\ \mu Y.(\exists y.\textsc{live}(y) \wedge Grad(x,y) \vee \langle - \rangle(\textsc{live}(x) \rightarrow Y)) \wedge [-]X)$$

instead states that, along every path, it is always true, for each student x, that there exists an evolution in which either x is not persisted, or becomes eventually graduated (with final mark y).

4.7.5 Mu-Calculus with Description Logic Queries

As we have already discussed, the KAB present at the Semantic Layer of an ACSI artifact-based system contains a knowledge component that provides a rich, conceptual description of the domain at hand. This is done by means of a TBox that provides a description of the concepts of interests and their relationships, using DL axioms. Consequently, it is highly desirable to be able to specifying complex dynamic properties that pose queries over such knowledge component.

This can be achieved by taking $\mu\mathcal{L}$ as presented before, changing the syntax and semantics of queries according to the chosen DL. First, queries must obey to the syntax of a query language that is suitable for such DL. Second, it is important to remember that the ontology conveys only incomplete information about the domain of interest, and therefore queries are answered under the certain answers semantics.

For example, with *DL-Lite*$_{\mathcal{A},id}$ one could adopt an extension of UCQs, called ECQs, which are queries of the query language *EQL-Lite*(UCQ) [40], that is, the FOL query language whose atoms are UCQs evaluated according to the certain answer semantics. An *ECQ* over a TBox T and an ABox A is a possibly open formula of the form (where q is a UCQ):

$$Q \quad ::= \quad [q] \;\mid\; [x = y] \;\mid\; \neg Q \;\mid\; Q_1 \wedge Q_2 \;\mid\; \exists x.Q$$

The semantics of the resulting $\mu\mathcal{L}$ variant is the same as the one shown in Figure 15, except for the extension of queries, which must be take into account also the KAB knowledge component $\mathcal{K}$ and becomes

$$(Q)^{\Upsilon}_{v,V} = \{s \in \Sigma \mid \textsc{ans}(Qv, \mathcal{K}, snap(s))\}$$

Given a boolean query Q, $\textsc{ans}(Q, \mathcal{K}, A)$ is the set of tuples of constants in the active domain of A (denoted by $\textsc{adom}(A)$) defined as follows:

$$\begin{aligned} \textsc{ans}([q], \mathcal{K}, A) &= cert(q, \mathcal{K}, A) \\ \textsc{ans}([x = y)], \mathcal{K}, A) &= \{(x,y) \in \textsc{adom}(A) \mid (\mathcal{K}, A) \models x{=}y\} \\ \textsc{ans}(\neg Q, T, A) &= \textsc{adom}(A) \setminus \textsc{ans}(Q, T, A) \\ \textsc{ans}(Q_1 \wedge Q_2, T, A) &= \textsc{ans}(Q_1, T, A) \cap \textsc{ans}(Q_2, T, A) \\ \textsc{ans}(\exists x.Q, T, A) &= \exists t \in \textsc{adom}(A).\textsc{ans}(Q_{x/t}, T, A) \end{aligned}$$

where $Q_{x/t}$ is the formula obtained from Q by substituting each occurrence of x with t.

Example 4.14 *An example of formula is:*

$$\exists x_1 \cdots \exists x_n. \bigwedge_{i \neq j} \neg[x_i = x_j] \quad \wedge \bigwedge_{i \in \{1,\ldots,n\}} \mu Z.([Stud(x_i)] \vee \langle - \rangle Z)$$

The formula asserts that there are at least n individuals occurring in the current ABox *, which are not known to be equal, and each of which eventually denotes a student along some execution path. Notice that the formula does not imply that all of these students will be in the same state, nor that they will all occur in a single run.*

4.7.6 Other Work

We close the section by reviewing some related literature which can provide useful hints on further technologies to explore.

Artifact-centric processes with no database.

Work on formal analysis of artifact-based business processes in restricted contexts has been reported in [79, 80, 18]. Properties investigated include reachability [79, 80], general temporal constraints [80], and the existence of complete execution or dead end [18]. For the variants considered in each paper, verification is generally undecidable; decidability results were obtained only under rather severe restrictions, e.g., restricting all pre-conditions to be "true" [79], restricting to bounded domains [80, 18], or restricting the pre- and post-conditions to be propositional, and thus not referring to data values [80]. [36] adopts an artifact model variation with arithmetic operations but no database. It proposes a criterion for comparing the expressiveness of specifications using the notion of *dominance*, based on the input/output pairs of business processes. Decidability relies on restricting runs to bounded length. [143] addresses the problem of the existence of a run that satisfies a temporal property, for a restricted case with no database and only propositional LTL properties. All of these works model no underlying database (and hence no integrity constraints).

Artifact-centric processes with underlying database.

More recently, two lines of work have considered artifact-centric processes that also model an underlying relational database. One considers branching time, one linear time.

Branching time.

One promising approach stems from a line of research that has started with [52] and continued with [12] and [13] in the context of artifact-centric processes. The connection between evolution of data-centric dynamic systems and data exchange that we exploit in this paper was first devised in [52]. There the dynamic system transition relation itself is described in terms of TGDs mapping the current state to the next, and the evolution of the system is essentially a form of chase. Under suitable weak acyclicity conditions such a chase terminates, thus making the transition system finite. A first-order μ-calculus without first-order quantification across states is used as the verification formalism for which decidability is shown. Notice the role of getting new objects/values from the external environment, played here by service calls, is played there by nulls. These ideas where further developed in [12], where TGDs where replaced by action rules with the same syntax as here. Semantically, however, the dynamic system formalism there is deeply different: what we call here service calls are treated there as uninterpreted Skolem terms. This results in an ad-hoc interpretation of equality which sees every Skolem term as equal only to itself (as in the case of nulls [52]). The same first-order μ-calculus without first-order quantification across states of [52] is used as the verification formalism, and a form of

weak acyclicity is used as a sufficient condition for getting finite-state transition systems and decidability.

Inspired by [12], [13] builds a similar framework where actions are specified via pre- and post-conditions given as FO formulae interpreted over active domains. The verification logic considered is a first-order variant of CTL with no quantification across states. Thus, it inherits the limitations discussed above on expressibility of liveness properties. In addition, the limited temporal expressivity of CTL precludes expressing certain desirable properties such as fairness. [13] shows that under the assumption that each state has a bounded active domain, one can construct an abstract finite transition system that can be checked instead of the original concrete transition system, which is infinite-state in general. The approach is similar to the one we developed independently for nondeterministic services, however without quantification across states, standard bisimilarity suffices. As opposed to our work, the decidability of checking state-boundedness is not investigated in [13], and no sufficient syntactic conditions are proposed.

Linear time.

The work in [64] considers an artifact model in which the infinite domain is equipped with a dense linear order, which can be mentioned in pre-, post-conditions, and properties. Runs can receive unbounded external input from an infinite domain. Verification is decidable even if the input accumulates in states, and runs are neither run-bounded, nor state-bounded. However, this expressive power requires restrictions that render the result incomparable to ours. First, the property language is a first-order extension of LTL, and it is shown that extension to branching time (CTL*) leads to undecidability. Second, the formulae in pre-, post-conditions and properties access read-only and read-write database relations differently, querying the latter only in limited fashion. In essence, data can be arbitrarily accumulated in read-write relations, but these can be queried only by checking that they contain a given tuple of constants. It is shown that this restriction is tight, as even the ability to check emptiness of a read-write relation leads to undecidability. In addition, no integrity constraints are supported as it is shown that allowing a single functional dependency leads to undecidability. [59] disallows read-write relations entirely (only the artifact variables are writable), but this allows the extension of the decidability result to integrity constraints expressed as embedded dependencies with terminating chase, and to any decidable arithmetic. Again the result is incomparable to ours, as our modeling needs include read-write relations and their unrestricted querying.

Beyond artifact-centric processes.

Static analysis for semantic web services is considered in [115], but in a context restricted to finite domains. More recently, [3] has studied automatic verification in the context of business processes based on Active XML documents.

The works [65, 132, 4] are ancestors of [64] from the context of verification of electronic commerce applications. Their models could conceptually (if not naturally) be encoded as artifact systems, but they correspond only to particular cases of the model in [64]. Also, they limit external inputs to essentially come from the active domain of the database, thus ruling out fresh values introduced during the run.

Infinite-state systems.

Artifact based systems are a particular case of infinite-state systems. Research on automatic verification of infinite-state systems has also focused on extending classical model checking techniques (e.g., see [30] for a survey). However, in much of this work the emphasis is on studying recursive control rather than data, which is either ignored or finitely abstracted. More recent work has been focusing specifically on data as a source of infinity. This includes augmenting recursive procedures with integer parameters [21], rewriting systems with data [20], Petri nets with data associated to tokens [107], automata and logics over infinite alphabets [23, 22, 117,

63, 102, 19, 20], and temporal logics manipulating data [63]. However, the restricted use of data and the particular properties verified have limited applicability to the business process setting we target here.

4.8 Semantic Log

In particular, we propose a modeling pattern that allows to take into account temporal evolutions at the ontological level, and thus represent properties of objects that may vary over time and retain all changes which such properties may undergo. Notice that this means somehow pushing the dynamics of the domain into the ontological representation, which is static in nature. The main advantage of using this technique in the ACSI paradigm is that it enables to promote at the Semantic Layer information that are generally stored in logs at the Artifact Layer, and consequently exploit automated reasoning techniques over them.

4.8.1 Ontologies and temporal information

While in real application domain the intensional knowledge represented by an ontology can be commonly considered stable, i.e., it does not change over time, we have to expect that the extensional knowledge is continuously affected by temporal evolution. This observation perfectly fits with the ACSI artifact paradigm, and in particular with the KAB dynamics as presented in Section 3.3. Consider for example an ontology representing the states that characterize an order of a product during its lifecycle (e.g., researched, assembling, assembled, etc.). In every time instant, the order is exactly in one state of this cycle. In this respect, two choices are possible: ignoring the temporal depth, or representing it in the ontology. The former choice implies that the ontology is able to provide information only about one state, i.e., the current one. This means that, when the state changes, the representation of the extensional level of the ontology is updated coherently, incorporating the information about the new state, while retaining all knowledge that is not affected by the change. In such a process the ontology is able to provide information about the current state of the domain, but it *forgets* all the information regarding previous states, and it is not possible to reconstruct the overall evolution of the system. Depending on the particular scenario, and the specific application at hand, such modeling limitation might be considered unacceptable. This motivates the research about new modeling approaches that allow overcoming such limitation, thus considering the possibility of representing the time into the ontology. In the ACSI setting, this is of key importance towards governance, reporting and analysis of an artifact-centric system, which are carried out at the Semantic Layer and typically require to consider not only the current information, but also its past evolution up to now.

4.8.2 Storing System Snapshots via Versioning

The most direct way to track changes of the domain modeled by the ontology is to store all information affected by the change with some kind of temporal information. In other words, such an approach requires that various versions of the instances of the ontology are maintained. An ontology version represents the state of the extensional information at a specific point in time. When the current instance of the ontology changes, the new version of the extensional information, resulting from such a change, is stored together with a *timestamp* label. Thus, an arbitrary number of changes can be maintained, each one that gives raise to a new timestamp-labeled ontology instance. While such an approach avoids loss of information and permits to design the domain of interest abstracting from the problem of state changes, on the other hand, it might require storing large amounts of information. Indeed, each version represents a complete instance for the ontology, independently from the amount of information that has really changed. In other words, for each change we are forced to memorize the whole extensional level of the ontology, reproducing also the information that has not been modified.

In the ACSI setting, this approach consists in accumulating, at the Semantic Layer, all the semantic snapshots of the system recognized so far. More specifically, every time the Artifact Layer performs a (macro)transition that leads to a new stable configuration of the data, the Semantic Layer does not only update accordingly via the application of the mappings, but it also keeps track of a timestamped copy of the previous semantic snapshots. Beside the huge amount of (redundant) information that needs to be recorded, this approach suffers of another drawback: it is bound to the granularity of the Artifact Layer, because the whole information present in the semantic log is extended in a "monolithic" way only when a (macro)transition is completed.

A simple yet comprehensive example of such behavior is illustrated in Figure 16. The example captures the evolution of an ontology representing the scenario on *Orders of products* presented above. In words, the intensional information specifies that *Orders* have two attributes: *orderID*, which indicates the identification number of an order, and *status*, which indicates the current state of the order within its lifecycle. Moreover, the ontology states that *Products* fulfill an *Orders*, and that they are made of (*madeOf*) *Components*. Initially, at time *T1*, we have that the *Order o1*, which has an ID equals to *324*, is in the state '*researched*'. Also, we know that the *Product p1* fulfills the order *o1*, and that it is made of the two components *c1* and *c2*. Suppose that the status of the order *o1* changes in time, passing from '*researched*' to '*assembling*' at time *T2*, and from '*assembling*' to '*assembled*' at time *T3*. In Figure 16, such three evolution steps are represented by three different timestamp-labeled versions of the ontology instance. Note that in each version it is memorized the whole extensional level of the ontology, and not only the information affected by the temporal evolution.

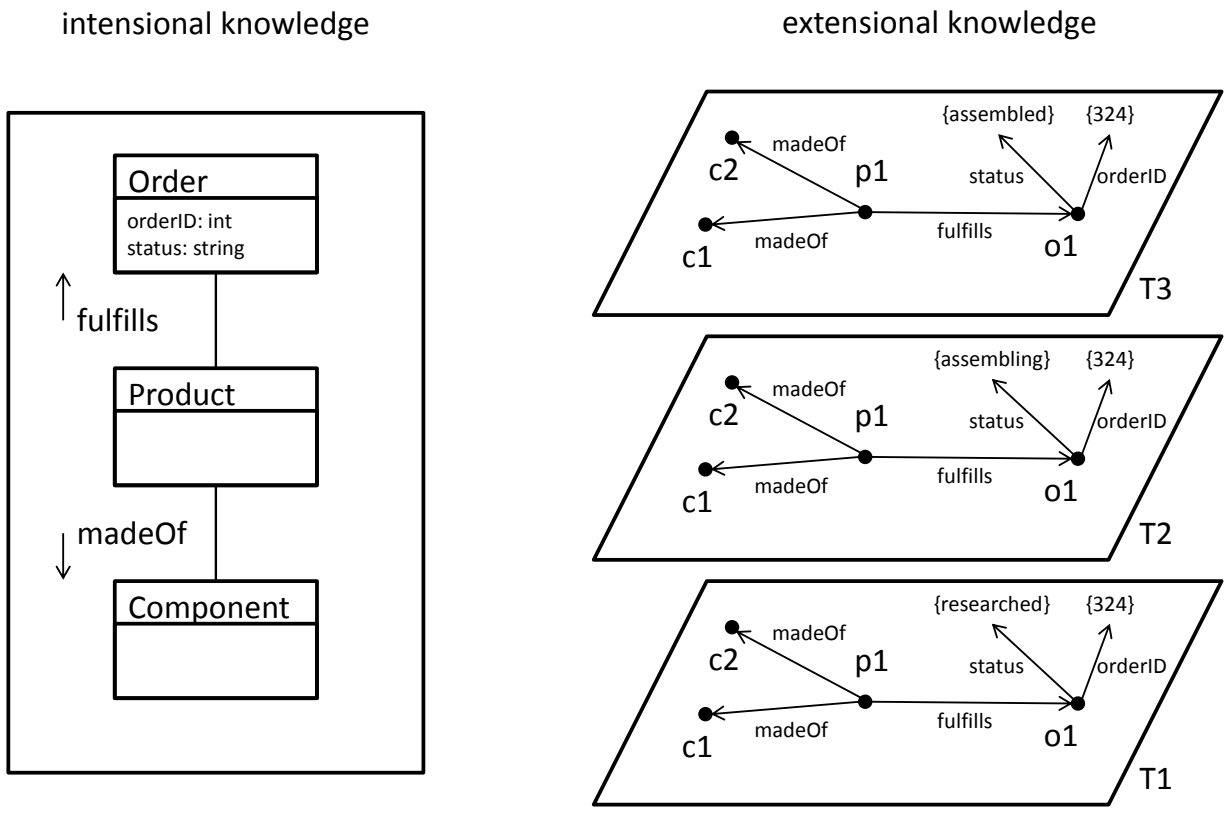


Figure 16 – Ontology evolution with versioning

4.8.3 Storing Local Changes via State Information

A different way to handle the evolution of the extensional level can be to introduce *the concept of state* in (the intensional level of) the ontology. The idea is to separate properties of objects that are considered not to change in time (or whose changes are not of interest) from properties of which we want to monitor the evolution. The first properties are also called *rigid* or *essential* in the literature [85, 84]. These represent those aspects that are inherent to the nature of object itself. In other terms, a change in these properties should compromise the identity of the object, and therefore is not considered.

Instead, the information that is affected by changes is modeled through properties of specific concepts of the ontology that represent states of the objects (which we call *state concepts* in the following). This modeling technique requires that an object o instance of a Concept C, which has changed over the time, has to be linked with various objects that are instances of the concept *State_Of_C*, which are the representations of o in the time line. The non-rigid properties of C are associated to the instance of the state concept *State_Of_C*. In addition, every instance of the concept *State_Of_C* is equipped with information about the period in which such information is valid. Note that, in such an approach identification constraints turn out to be an essential modeling construct. Indeed, a constraint that we want to impose is that there do not exist two instances of *State_Of_C* linked to the same instance o of C and with the same validity period, i.e., there are no two overlapping states for o.

Coming back to the example above, rigid information about the order, like the identification number *OrderID*, can be modeled as property of the class *Order*, while non-rigid information, like its status, is represented as a property of the class *StateOfOrder* (see Figure 17). In addition, the attribute *timestp* is associated to *StateOfOrder*. When an instance s of *StateOfOrder*, linked to an instance o of *Order* through the role *hasState*, is associated with a value v for *timestp*, s is considered valid from v to v', where v' is the timestamp of a different state s' of o such that $v' > v$ and there does not exist s'' for o with associated timestamp v'' such that $v' > v'' > v$. If such s' does not exist, s is considered currently valid (current state). Moreover, an identification constraint is specified which ensures that no two instances of *StateOfOrder* exist, which are linked to the same instance of order through the role *hasState*, and to the same value through the attribute *timestp*.

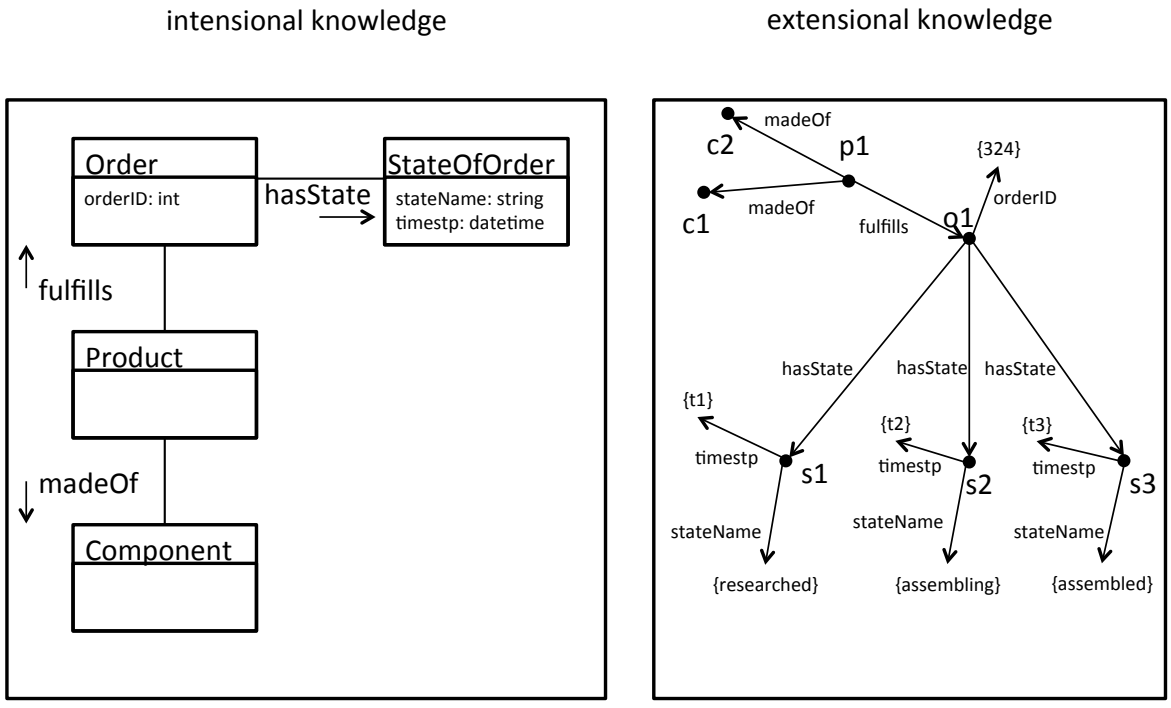


Figure 17 – Ontology evolution with states

A significant advantage of this approach is that one can treat changes of various concepts and properties with different granularity. For example, given a concept C, properties of C that change rarely may be associated to a state concept *StateA_Of_C*, whereas properties of C that change more often may be associated to the different state concept *StateB_Of_C*. Normally, an instance o of C will be linked to few instances of *StateA_Of_C* and many instances of *StateB_Of_C*, i.e., there exist few states for o with respect to the attributes of *StateA_Of_C* and many states with respect to the attributes of *StateB_Of_C*. Notice that using a single state concept in this case would have been disadvantageous, since this would have required to instantiate many properties in each state, often simply replicating the same values for them (for those properties that change rarely).

Another advantage is that it is possible to explicitly modeling the event that has caused the change of state (e.g., the assembly phase for a product is terminated and thus the product can be shipped), and relating through suitable roles such event to the state in which the event occurs (assembling phase) and to the state that is generated by the event (product assembled).

A disadvantage of this technique is that ontology designers are forced to consider aspects related to the changes during the design of the overall ontology.

4.8.4 Linkage to Logs at the Artifact Layer

Up to now, we have assumed that the log maintaining historical information about the execution of an artifact-centric system is encoded directly at the Semantic Layer. However, different design alternatives can be explored within the ACSI artifact paradigm.

For example, a log of the system's snapshots could be also maintained at the Artifact Layer. In this situation, the mappings could be exploited to obtain a corresponding semantic log at the Semantic Layer, with the advantage that such log can be queried and examined to check for higher level properties of the domain, or even meta properties of the Artifact Layer if they have been incorporated into the KAB knowledge component.

5 Order-to-Cash Scenario

The *order-to-cash* scenario describes a scenario in which a customer requests a generic product. An order for the product is created and submitted to an assembling company, which is responsible for collecting needed components from a set of suppliers. Once all components are received by the assembler, the product is fabricated and then shipped to the customer. During such a process, the customer may request the cancellation of the order.

In what follows we illustrate a simple formalization of this process in the guard-stage-milestone (GSM) formalism. In particular, we consider two artifact types, namely CustomerOrder, which is used for representing an order, and Component which represents for a single component.

Figures 18 and 19 show the lifecycle of CustomerOrder artifact type and the information model for both CustomerOrder and Component artifact types.

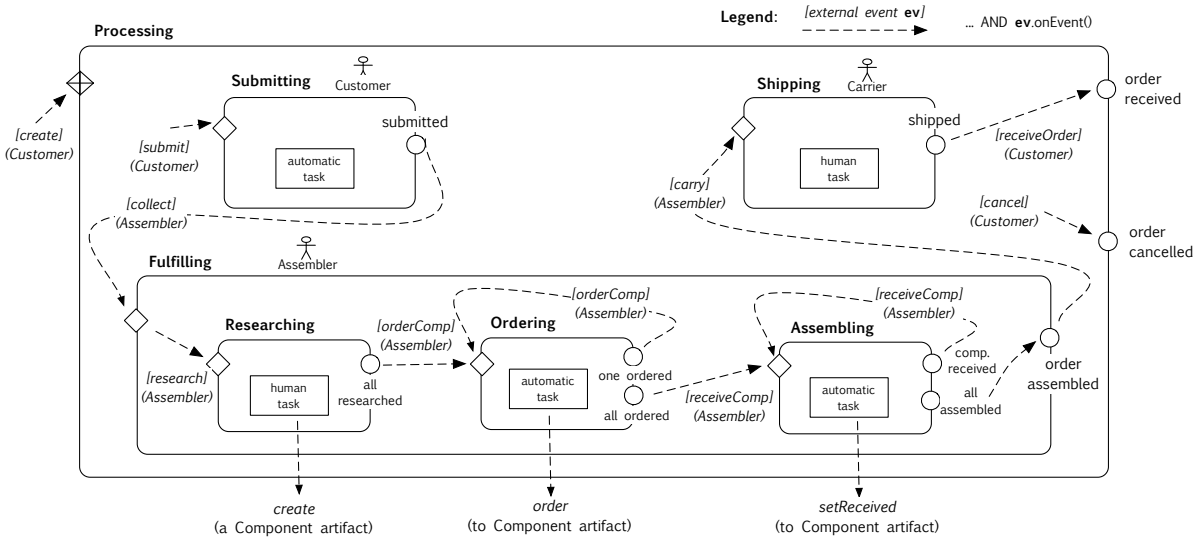


Figure 18 – GSM lifecycle of CustomerOrder **artifact**

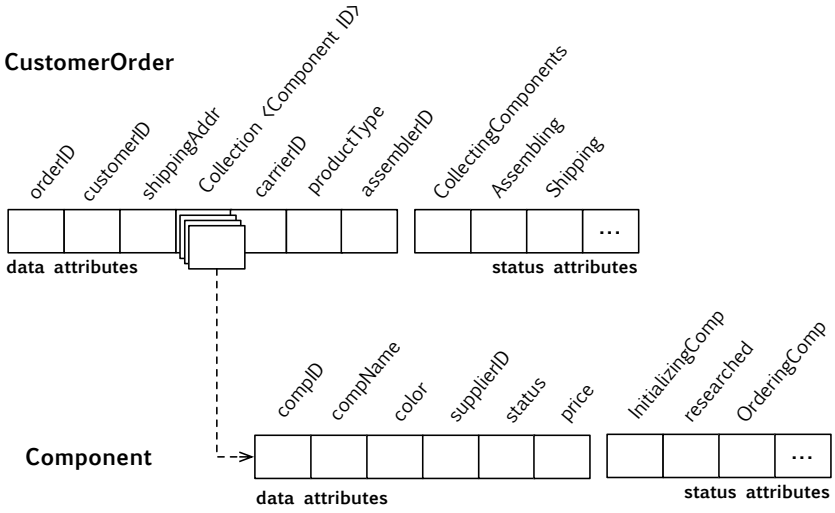


Figure 19 – GSM information model of CustomerOrder **and** Component **artifacts**

Analogously, Figure 20 shows the lifecycle of the Component artifact type.

At the beginning, an instance of the CustomerOrder artifact type is created when an event of type *create* is processed by the system. Such an event carries the desired product type and customer's ID and address. The customer then chooses an assembling company and submits the order by creating a *submit* event with the assembler ID as payload. The stage $OrderFullfillment$ allows the assembler to fabricate the product: the substage *Researching* is used to identify the

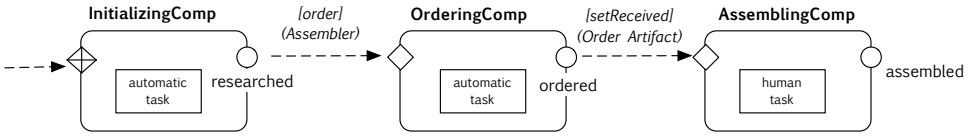


Figure 20 – GSM lifecycle of Component artifact

set of components required to build the product and, for each of them, a new *Component* artifact is created, filled with the information about the desired type of component (type name, price and color). As a result, milestone *researched* is eventually achieved for each of them, which is precisely the condition that makes milestone *all researched* achieved. At this point, components are ordered by the assembler to a supplier. This is done by generating an *orderComp* event (one for each of them), whose payload consists of a supplier ID. Such events causes *Ordering* substages to open in the *Order* artifact, also affecting the corresponding *Component* artifact, which is filled with the *supplierID* passed as payload. At this point, as an external event *receiveComp* (carrying a *compID*) is processed, *Assembling* stage opens and a *setReceived* event sent to the component artifact. In this way, components are received one by one (each time with a new *compID*), and eventually the *all assembled* milestone is achieved (*assembled* milestone is achieved for each component). Then, as an external *carry* event is processed (with a carrier ID as payload), the selected carrier is allowed to perform the shipping task, achieving *shipped* milestone. Finally, an external *receiveOrder* event is eventually generated by the customer and once processed, it signals that the product has been successfully shipped.

5.1 The Semantic Layer for the Order-to-Cash example

As already pointed out in Section 3, the semantic layer is a Knowledge and Action Based (KAB). A KAB is a tuple $\mathcal{KA} = (\mathcal{K}, \Gamma)$ where $\mathcal{K}$ is the *knowledge component* that focuses on the static aspects of the domain of interest, while Γ is the *action component*, characterizing the dynamic aspects of such a domain. In this Section we mostly concentrate on the knowledge component $\mathcal{K}$, that is composed by: *(i)* a TBox $\mathcal{T}$ (expressed in an ontology language based on Description Logic) that describes the conceptual representation of data, *(ii)* a set of mappings $\mathcal{M}$, linking such a conceptual layer with the Artifact Layer.

5.2 TBox

Figure 21 provides a graphical representation of a plausible TBox $\mathcal{T}$ for the order-to-cash scenario. Although in general the knowledge component provides a semantical representation of data of artifacts, events and gateways, here, for the sake of readability, we focus on data of artifacts only. The main concept is *Order*, which attributes are $orderID$ and *status*. We are interested in distinguishing whether an order is *processing*, *assembling*, *assembled* or *shipped*. An order is made for a certain *Product* that has a *ProductType*, and it is associated to a set of component types (*ComponentType* concept) that need to be assembled in order to build it. Of course, we require $madeOf$ role to be consistent with $composedOf$. Moreover, an order is linked to an instance of *Customer* concept by the *makesOrder* role. For each customer, we are interested in its social security number and address(es). Finally, other concepts of interest are the companies involved in the process. Concepts *Carrier* and *Assembler* are associated to *Order* to model the information about companies responsible for assembling and shipping the final product. An instance of concept *Supplier* is associated to each instance of concept *Component*, since each component used for assembling the product is provided by a possibly different supplier.

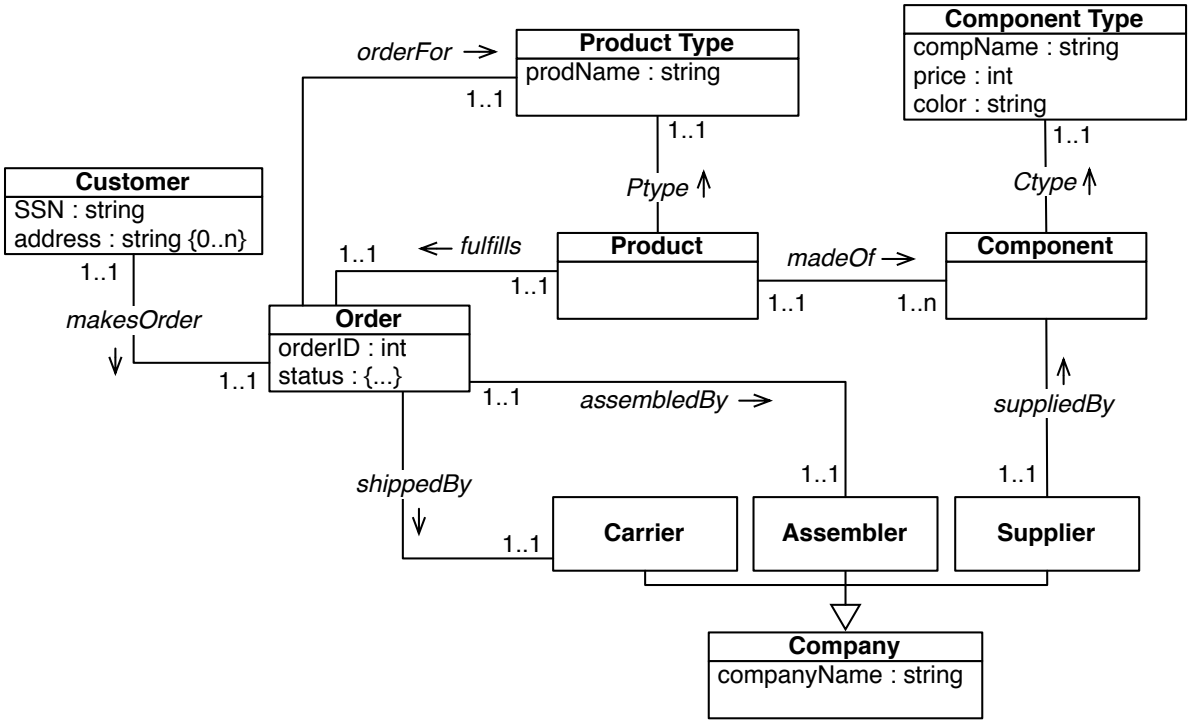


Figure 21 – Graphical representation of the TBox $\mathcal{T}$ for the order-to-cash example.

5.3 Mappings

For convenience, in order to define the set of mappings $\mathcal{M}$, we assume to have the following relational schema for the information model of the artifact types. Notice that such an assumption is trivial, since we can always reconstruct the data of each artifact by suitably querying this relational database. We mark with @ artifact identifiers.

Orders

orderID	productType	customerID	shippingAddress	AssemblerID	carrierID
@OrdArtifact1	ducati848	customer1	Park Avenue, NY	ducati	FedEx

Order_Status_Attributes

OrderID	CollectingComponents	Assembling	...
@OrdArtifact1	1	0	...

Components

compID	compName	supplierID	orderID
@CompArtifact1	frontWheel	metzler	@OrdArtifact1
@CompArtifact2	speedometer	gpt	@OrdArtifact1
@CompArtifact3	carbonExhaust	termignoni	@OrdArtifact1

Component_Types

compName	color	price
frontWheel	black	150
speedometer	red	200
carbonExhaust	matteBlack	300

Component_Status_Attributes

OrderID	InitializingComp	researched	...
@CompArtifact1	0	1	...
@CompArtifact2	0	1	...
@CompArtifact3	0	1	...

We now show how to maps data in artifacts' information models to concepts, roles and attributes of the ontology.

SELECT orderID, productType, customerID, shippingAddress, assemblerID FROM **Orders** $\rightsquigarrow$ {Order($ord(orderID)$), orderID($ord(orderID), orderID$)}

SELECT orderID, "processing" FROM **Order_Status_Attributes** WHERE allOrdered=0 $\rightsquigarrow$ status($ord(orderID)$, "processing")}

SELECT orderID, "assembling" FROM **Order_Status_Attributes** WHERE allOrdered=1 AND orderAssembled=0 $\rightsquigarrow$ status($ord(orderID)$, "assembling")}

SELECT orderID, "assembled" FROM **Order_Status_Attributes** WHERE orderAssembled=1 AND shipped=0 $\rightsquigarrow$ status($ord(orderID)$, "assembled")}

SELECT orderID, "shipped" FROM **Order_Status_Attributes** WHERE shipped=1 $\rightsquigarrow$ status($ord(orderID)$, "shipped")}

SELECT customerID, shippingAddress FROM **Order_Status_Attributes** $\rightsquigarrow$ {Customer($cust(customerID)$), ssn($cust(customerID), customerID$), address($cust(customerID), shippingAddress$)}

SELECT orderID, customerID FROM **Order_Status_Attributes** $\rightsquigarrow$ {makesOrder($cust(customerID), ord(orderID)$)}

SELECT productType FROM **Order_Status_Attributes** $\rightsquigarrow$ {ProductType($prodType(productType)$) prodName($prodType(productType), productType$)}

SELECT orderID, productType FROM **Order_Status_Attributes** $\rightsquigarrow$ {orderFor($ord(orderID), prodType(productType)$)}

SELECT orderID FROM **Order_Status_Attributes** $\rightsquigarrow$ {Product($prod(orderID)$), fulfills($prod(orderID), ord(orderID)$)}

SELECT orderID, productType FROM **Order_Status_Attributes** $\rightsquigarrow$ {Ptype($prodType(productType), prod(orderID)$)}

SELECT compID, orderID FROM **Components** $\rightsquigarrow$ {Component($comp(compID)$), madeOf($prod(orderID), comp(compID)$)}

SELECT compName, color, price FROM **Component_Types** $\rightsquigarrow$ {ComponentType($compType(compName), color, price$), compName($compType(compName), compName$) price($compType(compName), price$) color($compType(compName), color$)}

SELECT compID, compName FROM **Components** $\rightsquigarrow$ {Ctype($comp(compID), compType(compName)$)}

SELECT carrierID FROM **Order_Status_Attributes** $\rightsquigarrow$ {Carrier($carr(carrierID)$) companyID($carr(carrierID), carrierID$)}

SELECT orderID, carrierID FROM **Order_Status_Attributes** $\rightsquigarrow$ {shippedBy($ord(orderID), carr(carrierID)$)}

SELECT assemblerID FROM **Order_Status_Attributes** $\rightsquigarrow$ {Assembler($ass(assemblerID)$) companyID($ass(assemblerID), assemblerID$)}

SELECT orderID, assemblerID
FROM **Order_Status_Attributes** $\rightsquigarrow$ $\{\mathsf{assembledBy}(ord(orderID), ass(assemblerID))\}$

SELECT supplierID
FROM **Components** $\rightsquigarrow$ $\{\mathsf{Supplier}(supp(supplierID))$
$\mathsf{companyID}(supp(supplierID), supplierID)\}$

SELECT compID, supplierID
FROM **Components** $\rightsquigarrow$ $\{\mathsf{suppliedBy}(supp(supplierID), comp(compID))\}$

We briefly comment on such mappings, referring to Section 4.6 for the detailed discussion. The first mapping asserts that, for each Customer Order artifact ID in the data sources, i.e., the information model of Order artifacts, there is an Order concept instance with attribute $orderID$. Notice that, in order to cope with the impedance mismatch problem, each object appearing in the ontology is build by mean of a *functor*. In this case, the functor *ord* takes as parameter the (univocal) ID of an order. The attribute $orderID$, instead, makes the association between the object $ord(orderID)$ and its ID, namely $orderID$. The same intuitions hold for the other mappings.

5.4 Action Component

The action component Γ contains a set of dynamic laws, expressed at the Semantic Level, that define the behavioral boundaries that must be respected by processes and lifecycles defined in the Artifact Layer to reflect correctly the dynamics of the domain of interest. Such laws can be expressed in first order variants of temporal logics, such as CTL, LTL or μ-calculus. Here we present simple dynamic laws expressed informally in natural language, though the translation into a specific temporal logic is trivial. For instance, we may want to constraint possible system executions saying that it is not possible for a customer to cancel an order no matter how long it has been ordered. Indeed, if the order has been already assembled, hence it is ready to be shipped, the customer cannot cancel it. Such a law can be informally expressed as: *"In every state, if an order is assembled or shipped, it can neither be cancelled in the current state, nor in the future"*.

5.5 Semantic Traces

In this Section, we show a possible system evolution, of the of the order-to-cash example. We first consider a trace at the Artifact Layer, and then we focus on the corresponding semantic trace.

Recall that in GSM we have a new artifact snapshot anytime we process an external event, or a "stable" state is reached. In the following sequence of artifact snapshots, names on arrows represent the processed event (type).

$$s_0 \xrightarrow{create} s_1 \xrightarrow{submit} s_2 \longrightarrow s_3 \xrightarrow{collect} s_4 \xrightarrow{research} s_5 \longrightarrow s_6$$

We start from an initial snapshot s_0, and we show how the underlying data change when external events are processed. We list here the data of each snapshot, assuming the relational model above.

s_0 : *empty*

From s_0 we assume an event of type *create* is received, carrying, as payload, "ducati848" as product Type, and "Park ave." as shipping address. Others information are automatically generated by the system. Hence the next snapshot is:

s_1 : **Orders**(@OrdArtifact1,ducati848,customer1,Park Av., nil, nil)
@OrdArtifact1 stages = $\{Processing\}$

In s_1 an event of type *submit* is processed and it results in opening the *Submitting* stage of CustomerOrder artifact.

s_2 : **Orders**(@OrdArtifact1,ducati848,customer1,Park Av., ducati, nil)
@OrdArtifact1 stages = $\{Processing, Submitting\}$

Since stage *Submitting* is atomic, it performs its (automatic) task that, when completes, results in achieving the milestone, hence closing the stage. No external event is processed during this step.

s_3 : **Orders**(@OrdArtifact1,ducati848,customer1,Park Av., ducati, nil)
@OrdArtifact1 stages = $\{Processing\}$
@OrdArtifact1 milestones = $\{submitted\}$

When an event of type *collected* is processed, the $Fulfilling$ non-atomic stage opens.

s_4 : **Orders**{ (@OrdArtifact1,ducati848,customer1,Park Av., ducati, nil) }
@OrdArtifact1 stages = $\{Processing, Fulfilling\}$
@OrdArtifact1 milestones = $\{submitted\}$

Processing a *research* event results in opening the *Researching* stage. The task in it creates a *Component* artifact instance for each identified component in the payload of the event.

s_5 : **Orders**(@OrdArtifact1,ducati848,customer1,Park Av., ducati, nil)
Components(@CompArtifact1, frontWheel, metzler, @OrdArtifact1)
Components(@CompArtifact2, speedometer, gpt, @OrdArtifact1)
Components(@CompArtifact3, carbonExhaust, termigoni, @OrdArtifact1)
Component_Types(frontWheel, black, 150)
Component_Types(speedometer, red, 200)
Component_Types(carbonExhaust, matteBlack, 300)
@OrdArtifact1 stages = $\{Processing, Fulfilling, Researching\}$
@OrdArtifact1 milestones = $\{submitted\}$
@CompArtifact1 stages = $\{InitializingComp\}$
@CompArtifact2 stages = $\{InitializingComp\}$
@CompArtifact3 stages = $\{InitializingComp\}$

When the task completes, milestones *all researched* of CustomerOrder and milestones *reserached* of the Component instances just created are achieved, and hence the respective stages *Researching* close.

s_6 : **Orders**(@OrdArtifact1,ducati848,customer1,Park Av., ducati, nil)
Components(@CompArtifact1, frontWheel, metzler, @OrdArtifact1)
Components(@CompArtifact2, speedometer, gpt, @OrdArtifact1)
Components(@CompArtifact3, carbonExhaust, termigoni, @OrdArtifact1)
Component_Types(frontWheel, black, 150)
Component_Types(speedometer, red, 200)
Component_Types(carbonExhaust, matteBlack, 300)
@OrdArtifact1 stages = $\{Processing, Fulfilling\}$
@OrdArtifact1 milestones = $\{submitted, all\ researched\}$
@CompArtifact1 milestones = $\{researched\}$
@CompArtifact2 milestones = $\{researched\}$
@CompArtifact3 milestones = $\{researched\}$

We now show how the ontology evolves according to changes in the artifact layer. Indeed, by applying the mapping on each snapshot s_i, we obtain a corresponding ABox a_i. This implies a direct correspondance between each artifact snapshot and its semantic counterpart. Nonetheless, it may happen that multiple successive artifact snapshots map into the same semantic snapshot, i.e., same ABox, meaning that every change occurring at the artifact layer is not necessarily reflected at the semantic layer. The following sequence of ABoxes are simply obtain by applying the mappings to the (artifact) trace.

a_0 : *empty*

a_1 : Order(ord(@OrdArtifact1))
status(ord(@OrdArtifact1), "processing")
ProductType($prodType(ducati848)$)
prodName($prodType(ducati848), ducati848$)
orderFor(ord(@OrdArtifact1), $prodType(ducati848)$)
Product($prod$(@OrdArtifact1))
fulfills($prod$(@OrdArtifact1), ord(@OrdArtifact1))
Ptype($prod$(@OrdArtifact1), $prodType(ducati848)$)
Customer($cust(customer1)$)
ssn($cust(customer1), customer1$)
address($cust(customer1), Park\ Av.$)
makesOrder($cust(customer1)$, ord(@OrdArtifact1))

a_2 : Order(ord(@OrdArtifact1))
status(ord(@OrdArtifact1), "processing")
ProductType($prodType(ducati848)$)
prodName($prodType(ducati848), ducati848$)
orderFor(ord(@OrdArtifact1), $prodType(ducati848)$)
Product($prod$(@OrdArtifact1))
fulfills($prod$(@OrdArtifact1), ord(@OrdArtifact1))
Ptype($prod$(@OrdArtifact1), $prodType(ducati848)$)
Customer($cust(customer1)$)
ssn($cust(customer1, cust1$)
address($cust(customer1), Park\ Av.$)
makesOrder($cust(customer1)$, ord(@OrdArtifact1))
Assembler(ass(ducati))
companyName($ass(ducati), ducati$)
assembledBy(ord(@OrdArtifact1), $ass(ducati)$)

a_3 : Order(ord(@OrdArtifact1))
status(ord(@OrdArtifact1), "processing")
ProductType($prodType(ducati848)$)
prodName($prodType(ducati848), ducati848$)
orderFor(ord(@OrdArtifact1), $prodType(ducati848)$)
Product($prod$(@OrdArtifact1))
fulfills($prod$(@OrdArtifact1), ord(@OrdArtifact1))
Ptype($prod$(@OrdArtifact1), $prodType(ducati848)$)
Customer($cust(customer1)$)
ssn($cust(customer1), customer1$)
address($cust(customer1), Park\ Av.$)
makesOrder($cust(customer1)$, ord(@OrdArtifact1))
Assembler($ass(ducati)$)
companyName($ass(ducati), ducati$)
assembledBy(ord(@OrdArtifact1), $ass(ducati)$)

ComponentType($compType(frontWheel)$)
compName($compType(frontWheel), frontWheel$)
price($compType(frontWheel), 150$)
color($compType(frontWheel), black$)
ComponentType($compType(speedometer)$)
compName($compType(speedometer), speedometer$)
price($compType(speedometer), 200$)
color($compType(speedometer), red$)
ComponentType($compType(carbonExhaust)$)
compName($compType(carbonExhaust), carbonExhaust$)
price($compType(carbonExhaust), 300$)
color($compType(carbonExhaust), matteBlack$)
Component($comp$(@CompArtifact1))
Component($comp$(@CompArtifact2))
Component($comp$(@CompArtifact3))
Ctype($comp$(@CompArtifact1), $compType(frontWheel)$)
Ctype($comp$(@CompArtifact2), $compType(speedometer)$)
Ctype($comp$(@CompArtifact3), $compType(carbonExhaust)$)
madeOf($prod$(@OrdArtifact1), $comp$(@CompArtifact1))
madeOf($prod$(@OrdArtifact1), $comp$(@CompArtifact2))
madeOf($prod$(@OrdArtifact1), $comp$(@CompArtifact3))

Notice that, as expected, artifact snapshots s_2, s_3 and s_4 maps to the same semantic snapshot a_2. The same happens to s_5 and s_6 that maps to a_3.

Moreover, it is easy to check that such a semantic trace does verify the dynamic law in the action component, since, trivially, order @order1 is never cancelled.

References

[1] S. Abiteboul and O. Duschka. Complexity of answering queries using materialized views. In *Proc. of the 17th ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS'98)*, pages 254–265, 1998.

[2] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison Wesley Publ. Co., 1995.

[3] S. Abiteboul, L. Segoufin, and V. Vianu. Static analysis of active XML systems. *ACM Trans. Database Syst.*, 34(4), 2009.

[4] S. Abiteboul, V. Vianu, B. Fordham, and Y. Yesha. Relational transducers for electronic commerce. In *J. of Computer and System Sciences*, pages 179–187, 1998.

[5] M. Arenas, J. Pérez, J. L. Reutter, and C. Riveros. Foundations of schema mapping management. In *Proc. of the 29th ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS 2010)*, pages 227–238, 2010.

[6] A. Artale, D. Calvanese, R. Kontchakov, and M. Zakharyaschev. DL-Lite in the light of first-order logic. In *Proc. of the 22nd AAAI Conf. on Artificial Intelligence (AAAI 2007)*, pages 361–366, 2007.

[7] A. Artale, D. Calvanese, R. Kontchakov, and M. Zakharyaschev. The *DL-Lite* family and relations. *J. of Artificial Intelligence Research*, 36:1–69, 2009.

[8] F. Baader. Augmenting concept languages by transitive closure of roles: An alternative to terminological cycles. In *Proc. of the 12th Int. Joint Conf. on Artificial Intelligence (IJCAI'91)*, 1991.

[9] F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation and Applications*. Cambridge University Press, 2003.

[10] F. Baader, J. Hladik, C. Lutz, and F. Wolter. From tableaux to automata for description logics. *Fundamenta Informaticae*, 57:1–33, 2003.

[11] F. Baader and U. Sattler. An overview of tableau algorithms for description logics. *Studia Logica*, 69(1):5–40, 2001.

[12] B. Bagheri Hariri, D. Calvanese, G. De Giacomo, R. De Masellis, and P. Felli. Foundations of relational artifacts verification. In *BPM*, 2011.

[13] F. Belardinelli, A. Lomuscio, and F. Patrizi. Verification of deployed artifact systems via data abstraction. In *ICSOC*, 2011.

[14] M. Ben-Ari, A. Pnueli, and Z. Manna. The Temporal Logic of Branching Time. *Acta Informatica*, 20:207–226, 1983.

[15] D. Beneventano, S. Bergamaschi, S. Castano, A. Corni, R. Guidetti, G. Malvezzi, M. Melchiori, and M. Vincini. Information integration: the MOMIS project demonstration. In *Proc. of the 26th Int. Conf. on Very Large Data Bases (VLDB 2000)*, 2000.

[16] P. A. Bernstein, F. Giunchiglia, A. Kementsietsidis, J. Mylopoulos, L. Serafini, and I. Zaihrayeu. Data management for peer-to-peer computing: A vision. In *Proc. of the 5th Int. Workshop on the Web and Databases (WebDB 2002)*, 2002.

[17] K. Bhattacharya, N. S. Caswell, S. Kumaran, A. Nigam, and F. Y. Wu. Artifact-centered operational modeling: Lessons from customer engagements. *IBM Systems Journal*, 46(4):703–721, 2007.

[18] K. Bhattacharya, C. E. Gerede, R. Hull, R. Liu, and J. Su. Towards formal analysis of artifact-centric business process models. In *BPM*, 2007.

[19] M. Bojanczyk, A. Muscholl, T. Schwentick, L. Segoufin, and C. David. Two-variable logic on words with data. In *Proc. of the 21st IEEE Symp. on Logic in Computer Science (LICS 2006)*, pages 7–16, 2006.

[20] A. Bouajjani, P. Habermehl, Y. Jurski, and M. Sighireanu. Rewriting systems with data. In *FCT'07*, 2007.

[21] A. Bouajjani, P. Habermehl, and R. Mayr. Automatic verification of recursive procedures with one integer parameter. *Theoretical Computer Science*, 295, 2003.

[22] P. Bouyer. A logical characterization of data languages. *Information Processing Lett.*, 84(2), 2002.

[23] P. Bouyer, A. Petit, and D. Thérien. An algebraic approach to data languages and timed languages. *Information and Computation*, 182(2), 2003.

[24] R. J. Brachman and H. J. Levesque. The tractability of subsumption in frame-based description languages. In *Proc. of the 4th Nat. Conf. on Artificial Intelligence (AAAI'84)*, pages 34–37, 1984.

[25] R. J. Brachman and H. J. Levesque, editors. *Readings in Knowledge Representation*. Morgan Kaufmann, 1985.

[26] R. J. Brachman and J. G. Schmolze. An overview of the KL-ONE knowledge representation system. *Cognitive Science*, 9(2):171–216, 1985.

[27] J. Bradfield and C. Stirling. Modal mu-calculi. In *Handbook of Modal Logic*, volume 3, pages 721–756. Elsevier, 2007.

[28] M. L. Brodie, J. Mylopoulos, and J. W. Schmidt, editors. *On Conceptual Modeling: Perspectives from Artificial Intelligence, Databases, and Programming Languages*. Springer, 1984.

[29] M. Buchheit, F. M. Donini, and A. Schaerf. Decidable reasoning in terminological knowledge representation systems. *J. of Artificial Intelligence Research*, 1:109–138, 1993.

[30] O. Burkart, D. Caucal, F. Moller, and B. Steffen. Verification of infinite structures. In *Handbook of Process Algebra*. Elsevier Science, 2001.

[31] A. Calì, D. Calvanese, G. De Giacomo, and M. Lenzerini. Accessing data integration systems through conceptual schemas. In *Proc. of the 20th Int. Conf. on Conceptual Modeling (ER 2001)*, volume 2224 of *Lecture Notes in Computer Science*, pages 270–284. Springer, 2001.

[32] A. Calì, D. Calvanese, G. De Giacomo, and M. Lenzerini. On the expressive power of data integration systems. In *Proc. of the 21st Int. Conf. on Conceptual Modeling (ER 2002)*, volume 2503 of *Lecture Notes in Computer Science*, pages 338–350. Springer, 2002.

[33] A. Calì, D. Calvanese, G. De Giacomo, and M. Lenzerini. Data integration under integrity constraints. *Information Systems*, 29:147–163, 2004.

[34] A. Calì, D. Calvanese, G. De Giacomo, M. Lenzerini, P. Naggar, and F. Vernacotola. IBIS: Semantic data integration at work. In *Proc. of the 15th Int. Conf. on Advanced Information Systems Engineering (CAiSE 2003)*, pages 79–94, 2003.

[35] D. Calvanese and G. De Giacomo. Expressive description logics. In Baader et al. [9], chapter 5, pages 178–218.

[36] D. Calvanese, G. De Giacomo, R. Hull, and J. Su. Artifact-centric workflow dominance. In *ICSOC/ServiceWave*, 2009.

[37] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, A. Poggi, M. Rodriguez-Muro, R. Rosati, M. Ruzzi, and D. F. Savo. The Mastro system for ontology-based data access. *Semantic Web J.*, 2(1):43–53, 2011.

[38] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, A. Poggi, R. Rosati, and M. Ruzzi. Data integration through *DL-Lite*$_\mathcal{A}$ ontologies. In K.-D. Schewe and B. Thalheim, editors, *Revised Selected Papers of the 3rd Int. Workshop on Semantics in Data and Knowledge Bases (SDKB 2008)*, volume 4925 of *Lecture Notes in Computer Science*, pages 26–47. Springer, 2008.

[39] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, and R. Rosati. Data complexity of query answering in description logics. In *Proc. of the 10th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR 2006)*, pages 260–270, 2006.

[40] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, and R. Rosati. EQL-Lite: Effective first-order query processing in description logics. In *Proc. of the 20th Int. Joint Conf. on Artificial Intelligence (IJCAI 2007)*, pages 274–279, 2007.

[41] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, and R. Rosati. Tractable reasoning and efficient query answering in description logics: The *DL-Lite* family. *J. of Automated Reasoning*, 39(3):385–429, 2007.

[42] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, and R. Rosati. Path-based identification constraints in description logics. In *Proc. of the 11th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR 2008)*, pages 231–241, 2008.

[43] D. Calvanese, G. De Giacomo, and M. Lenzerini. Reasoning in expressive description logics with fixpoints based on automata on infinite trees. In *Proc. of the 16th Int. Joint Conf. on Artificial Intelligence (IJCAI'99)*, pages 84–89, 1999.

[44] D. Calvanese, G. De Giacomo, and M. Lenzerini. Answering queries using views over description logics knowledge bases. In *Proc. of the 17th Nat. Conf. on Artificial Intelligence (AAAI 2000)*, pages 386–391, 2000.

[45] D. Calvanese, G. De Giacomo, and M. Lenzerini. 2ATAs make DLs easy. In *Proc. of the 15th Int. Workshop on Description Logic (DL 2002)*, volume 53 of *CEUR Electronic Workshop Proceedings,* `http://ceur-ws.org/`, pages 107–118, 2002.

[46] D. Calvanese, G. De Giacomo, and M. Lenzerini. Description logics for information integration. In A. Kakas and F. Sadri, editors, *Computational Logic: Logic Programming and Beyond, Essays in Honour of Robert A. Kowalski*, volume 2408 of *Lecture Notes in Computer Science*, pages 41–60. Springer, 2002.

[47] D. Calvanese, G. De Giacomo, M. Lenzerini, and D. Nardi. Reasoning in expressive description logics. In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, volume II, chapter 23, pages 1581–1634. Elsevier Science Publishers, 2001.

[48] D. Calvanese, G. De Giacomo, M. Lenzerini, D. Nardi, and R. Rosati. Data integration in data warehousing. *Int. J. of Cooperative Information Systems*, 10(3):237–271, 2001.

[49] D. Calvanese, G. De Giacomo, M. Lenzerini, and M. Y. Vardi. Containment of conjunctive regular path queries with inverse. In *Proc. of the 7th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR 2000)*, pages 176–185, 2000.

[50] D. Calvanese, E. Kharlamov, W. Nutt, and D. Zheleznyakov. Updating ABoxes in *DL-Lite*. In *Proc. of the 4th Alberto Mendelzon Int. Workshop on Foundations of Data Management (AMW 2010)*, volume 619 of *CEUR Electronic Workshop Proceedings,* `http://ceur-ws.org/`, pages 3.1–3.12, 2010.

[51] D. Calvanese, M. Lenzerini, and D. Nardi. Unifying class-based representation formalisms. *J. of Artificial Intelligence Research*, 11:199–240, 1999.

[52] P. Cangialosi, G. De Giacomo, R. De Masellis, and R. Rosati. Conjunctive artifact-centric services. In *Proc. ICSOC*, 2010.

[53] M. J. Carey, L. M. Haas, P. M. Schwarz, M. Arya, W. F. Cody, R. Fagin, M. Flickner, A. Luniewski, W. Niblack, D. Petkovic, J. Thomas, J. H. Williams, and E. L. Wimmers. Towards heterogeneous multimedia information systems: The Garlic approach. In *Proc. of the 5th Int. Workshop on Research Issues in Data Engineering – Distributed Object Management (RIDE-DOM'95)*, pages 124–131. IEEE Computer Society Press, 1995.

[54] C. Chen, V. Haarslev, and J. Wang. LAS: Extending Racer by a Large ABox Store. In *Proc. of the 18th Int. Workshop on Description Logic (DL 2005)*, volume 147 of *CEUR Electronic Workshop Proceedings,* `http://ceur-ws.org/`, 2005.

[55] E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic Verification of Finite-State Concurrent Systems Using Temporal Logic Specifications. *ACM Transactions on Programming Languages and Systems*, 8(2):244–263, 1986.

[56] E. M. Clarke, O. Grumberg, and D. A. Peled. *Model checking*. The MIT Press, Cambridge, MA, USA, 1999.

[57] E. M. Clarke and J. M. Wing. Formal Methods: State of the Art and Future Directions. *ACM Computing Surveys*, 28(4):626–643, 1996.

[58] M. Dam. CTL* and ECTL* as Fragments of the Modal *mu*-Calculus. *Theoretical Computure Science*, 126(1):77–96, 1994.

[59] E. Damaggio, A. Deutsch, and V. Vianu. Artifact systems with data dependencies and arithmetic. In *14th International Conference on Database Theory (ICDT 2011)*, pages 66–77. ACM, 2011.

[60] G. De Giacomo, M. Lenzerini, A. Poggi, and R. Rosati. On instance-level update and erasure in description logic ontologies. *J. of Logic and Computation, Special Issue on Ontology Dynamics*, 19(5):745–770, 2009.

[61] G. De Giacomo, Y. Lespérance, and H. J. Levesque. ConGolog, a concurrent programming language based on the situation calculus. *Artificial Intelligence*, 121(1–2):109–169, 2000.

[62] S. Decker, M. Erdmann, D. Fensel, and R. Studer. Ontobroker: Ontology based access to distributed and semi-structured information. In R. Meersman, Z. Tari, and S. Stevens, editors, *Database Semantic: Semantic Issues in Multimedia Systems*, chapter 20, pages 351–370. Kluwer Academic Publishers, 1999.

[63] S. Demri and R. Lazić. LTL with the freeze quantifier and register automata. *ACM Transactions on Computational Logic*, 10, 2009.

[64] A. Deutsch, R. Hull, F. Patrizi, and V. Vianu. Automatic verification of data-centric business processes. In *12th International Conference on Database Theory (ICDT 2009)*. ACM, 2009.

[65] A. Deutsch, L. Sui, and V. Vianu. Specification and verification of data-driven web applications. *JCSS*, 73(3):442–474, 2007.

[66] F. M. Donini, M. Lenzerini, D. Nardi, and W. Nutt. The complexity of concept languages. *Information and Computation*, 134:1–58, 1997.

[67] O. Duschka. *Query Planning and Optimization in Information Integration.* PhD thesis, Stanford University, 1997.

[68] O. Duschka and M. R. Genesereth. Infomaster – an information integration tool. In *Proceedings of the International Workshop on Intelligent Information Integration during the 21st German Annual Conference on Artificial Intelligence (KI'97)*, 1997.

[69] C. Eisner and D. Fisman. *A Practical Introduction to PSL.* Integrated Circuits and Systems. Springer, 2006.

[70] E. A. Emerson. Model checking and the Mu-calculus. In N. Immerman and P. Kolaitis, editors, *Proc. of the DIMACS Symposium on Descriptive Complexity and Finite Model*, pages 185–214. American Mathematical Society Press, 1997.

[71] E. A. Emerson and E. M. Clarke. Characterizing correctness properties of parallel programs using fixpoints. In *Proc. of the 7th Coll. on Automata, Languages and Programming (ICALP'80)*, pages 169–181, 1980.

[72] E. A. Emerson and J. Y. Halpern. "Sometimes" and "Not Never" revisited: on branching versus linear time temporal logic. *Journal of the ACM*, 33(1):151–178, 1986.

[73] R. Fagin, P. G. Kolaitis, R. J. Miller, and L. Popa. Data exchange: Semantics and query answering. In *Proc. of the 9th Int. Conf. on Database Theory (ICDT 2003)*, pages 207–224, 2003.

[74] R. Fagin, P. G. Kolaitis, R. J. Miller, and L. Popa. Data exchange: Semantics and query answering. *Theoretical Computer Science*, 336(1):89–124, 2005.

[75] R. Fagin, P. G. Kolaitis, and L. Popa. Data exchange: Getting to the core. In *Proc. of the 22nd ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS 2003)*, pages 90–101, 2003.

[76] R. Fagin, P. G. Kolaitis, L. Popa, and W.-C. Tan. Quasi-inverses of schema mappings. *ACM Trans. on Database Systems*, 33(2):1–52, 2008.

[77] M. Friedman, A. Levy, and T. Millstein. Navigational plans for data integration. In *Proc. of the 16th Nat. Conf. on Artificial Intelligence (AAAI'99)*, pages 67–73. AAAI Press/The MIT Press, 1999.

[78] H. Garcia-Molina, Y. Papakonstantinou, D. Quass, A. Rajaraman, Y. Sagiv, J. D. Ullman, V. Vassalos, and J. Widom. The TSIMMIS approach to mediation: Data models and languages. *J. of Intelligent Information Systems*, 8(2):117–132, 1997.

[79] C. E. Gerede, K. Bhattacharya, and J. Su. Static analysis of business artifact-centric operational models. In *IEEE International Conference on Service-Oriented Computing and Applications*, 2007.

[80] C. E. Gerede and J. Su. Specification and verification of artifact behaviors in business process models. In *ICSOC*, 2007.

[81] F. Goasdoue, V. Lattes, and M.-C. Rousset. The use of CARIN language and algorithms for information integration: The Picsel system. *Int. J. of Cooperative Information Systems*, 9(4):383–401, 2000.

[82] T. Gruber. Towards principles for the design of ontologies used for knowledge sharing. *Int. J. of Human and Computer Studies*, 43(5/6):907–928, 1995.

[83] T. R. Gruber. A translation approach to portable ontology specification. *Knowledge Acquisition*, 5(2):199–220, 1993.

[84] N. Guarino and C. A. Welty. Ontological analysis of taxonomic relationships. In *Proc. of the 19th Int. Conf. on Conceptual Modeling (ER 2000)*, pages 210–224, 2000.

[85] N. Guarino and C. A. Welty. An overview of OntoClean. In S. Staab and R. Studer, editors, *Handbook on Ontologies*. Springer, 2004.

[86] V. Haarslev and R. Möller. RACER system description. In *Proc. of the Int. Joint Conf. on Automated Reasoning (IJCAR 2001)*, volume 2083 of *Lecture Notes in Artificial Intelligence*, pages 701–705. Springer, 2001.

[87] A. Y. Halevy. Answering queries using views: A survey. *Very Large Database J.*, 10(4):270–294, 2001.

[88] A. Y. Halevy. Data integration: A status report. In *Proc. of the 10th Conf. on Database Systems for Business, Technology and Web (BTW 2003)*, pages 24–29, 2003.

[89] A. Y. Halevy, A. Rajaraman, and J. Ordille. Data integration: The teenage years. In *Proc. of the 32nd Int. Conf. on Very Large Data Bases (VLDB 2006)*, pages 9–16, 2006.

[90] D. Harel, D. Kozen, and J. Tiuryn. *Dynamic Logic*. The MIT Press, 2000.

[91] J. Heflin and J. Hendler. A portrait of the Semantic Web in action. *IEEE Intelligent Systems*, 16(2):54–59, 2001.

[92] S. Heymans, L. Ma, D. Anicic, Z. Ma, N. Steinmetz, Y. Pan, J. Mei, A. Fokoue, A. Kalyanpur, A. Kershenbaum, E. Schonberg, K. Srinivas, C. Feier, G. Hench, B. Wetzstein, and U. Keller. Ontology reasoning with large data repositories. In M. Hepp, P. De Leenheer, A. de Moor, and Y. Sure, editors, *Ontology Management, Semantic Web, Semantic Web Services, and Business Applications*, volume 7 of *Semantic Web And Beyond Computing for Human Experience*, pages 89–128. Springer, 2008.

[93] I. Horrocks. Using an expressive description logic: FaCT or fiction? In *Proc. of the 6th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR'98)*, pages 636–647, 1998.

[94] I. Horrocks, L. Li, D. Turi, and S. Bechhofer. The Instance Store: DL reasoning with large numbers of individuals. In *Proc. of the 17th Int. Workshop on Description Logic (DL 2004)*, volume 104 of *CEUR Electronic Workshop Proceedings,* `http://ceur-ws.org/`, 2004.

[95] I. Horrocks, P. F. Patel-Schneider, and F. van Harmelen. From $\mathcal{SHIQ}$ and RDF to OWL: The making of a web ontology language. *J. of Web Semantics*, 1(1):7–26, 2003.

[96] I. Horrocks and U. Sattler. A description logic with transitive and inverse roles and role hierarchies. *J. of Logic and Computation*, 9(3):385–410, 1999.

[97] I. Horrocks and U. Sattler. A tableau decision procedure for $\mathcal{SHOIQ}$. *J. of Automated Reasoning*, 39(3):249–276, 2007.

[98] R. Hull. A survey of theoretical research on typed complex database objects. In J. Paredaens, editor, *Databases*, pages 193–256. Academic Press, 1988.

[99] R. Hull. Managing semantic heterogeneity in databases: A theoretical perspective. In *Proc. of the 16th ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS'97)*, pages 51–61, 1997.

[100] U. Hustadt, B. Motik, and U. Sattler. Data complexity of reasoning in very expressive description logics. In *Proc. of the 19th Int. Joint Conf. on Artificial Intelligence (IJCAI 2005)*, pages 466–471, 2005.

[101] T. Imielinski and W. L. Jr. Incomplete information in relational databases. *J. of the ACM*, 31(4):761–791, 1984.

[102] M. Jurdzinski and R. Lazić. Alternation-free modal mu-calculus for data trees. In *22nd IEEE Symposium on Logic in Computer Science (LICS 2007)*, 2007.

[103] T. Kirk, A. Y. Levy, Y. Sagiv, and D. Srivastava. The Information Manifold. In *Proceedings of the AAAI 1995 Spring Symp. on Information Gathering from Heterogeneous, Distributed Enviroments*, pages 85–91, 1995.

[104] P. G. Kolaitis. Schema mappings, data exchange, and metadata management. In *Proc. of the 24rd ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS 2005)*, pages 61–75, 2005.

[105] D. Kozen. Results on the Propositional mu-Calculus. *Theoretical Computoter Science*, 27:333–354, 1983.

[106] A. Krisnadhi and C. Lutz. Data complexity in the $\mathcal{EL}$ family of description logics. In *Proc. of the 14th Int. Conf. on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR 2007)*, pages 333–347, 2007.

[107] R. Lazić, T. Newcomb, J. Ouaknine, A. Roscoe, and J. Worrell. Nets with tokens which carry data. In *ICATPN'07*, 2007.

[108] M. Lenzerini. Data integration: A theoretical perspective. In *Proc. of the 21st ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS 2002)*, pages 233–246, 2002.

[109] A. Y. Levy. Logic-based techniques in data integration. In J. Minker, editor, *Logic Based Artificial Intelligence*. Kluwer Academic Publishers, 2000.

[110] L. Libkin. Data exchange and incomplete information. In *Proc. of the 25th ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS 2006)*, pages 60–69, 2006.

[111] Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems: Specification*. Springer, 1992.

[112] I. Manolescu, D. Florescu, and D. Kossmann. Answering XML queries on heterogeneous data sources. In *Proc. of the 27th Int. Conf. on Very Large Data Bases (VLDB 2001)*, pages 241–250, 2001.

[113] M. Minsky. A framework for representing knowledge. In J. Haugeland, editor, *Mind Design*. The MIT Press, 1981. A longer version appeared in *The Psychology of Computer Vision* (1975). Republished in [25].

[114] R. Möller and V. Haarslev. Description logic systems. In Baader et al. [9], chapter 8, pages 282–305.

[115] S. Narayanan and S. McIlraith. Simulation, verification and automated composition of web services. In *Intl. World Wide Web Conf. (WWW2002)*, 2002.

[116] B. Nebel. Computational complexity of terminological reasoning in BACK. *Artificial Intelligence*, 34(3):371–383, 1988.

[117] F. Neven, T. Schwentick, and V. Vianu. Finite State Machines for Strings Over Infinite Alphabets. *ACM Trans. on Computational Logic*, 5(3), 2004.

[118] N. F. Noy. Semantic integration: A survey of ontology-based approaches. *SIGMOD Record*, 33(4):65–70, 2004.

[119] M. Ortiz, D. Calvanese, and T. Eiter. Data complexity of query answering in expressive description logics via tableaux. *J. of Automated Reasoning*, 41(1):61–98, 2008.

[120] L. Palopoli, G. Terracina, and D. Ursino. The system DIKE: Towards the semi-automatic synthesis of cooperative information systems and data warehouses. In *Proc. of Symposium on Advances in Databases and Information Systems (ADBIS-DASFAA 2000)*, pages 108–117, 2000.

[121] D. M. R. Park. Finiteness is mu-ineffable. *Theoretical Computer Science*, 3(2), 1976.

[122] P. F. Patel-Schneider, D. L. McGuinness, R. J. Brachman, L. A. Resnick, and A. Borgida. The CLASSIC knowledge representation system: Guiding principles and implementation rational. *SIGART Bull.*, 2(3):108–113, 1991.

[123] A. Pnueli. A Temporal Logic of Concurrent Programs. *Theoretical Computer Science*, 13:45–60, 1981.

[124] A. Poggi, D. Lembo, D. Calvanese, G. De Giacomo, M. Lenzerini, and R. Rosati. Linking data to ontologies. *J. on Data Semantics*, X:133–173, 2008.

[125] R. Reiter. *Knowledge in Action: Logical Foundations for Specifying and Implementing Dynamical Systems*. The MIT Press, 2001.

[126] E. Sandewall. *Features and Fluents: The Representation of Knowledge about Dynamical Systems*. Clarendon Press, Oxford, 1994.

[127] K. Schild. A correspondence theory for terminological logics: Preliminary report. In *Proc. of the 12th Int. Joint Conf. on Artificial Intelligence (IJCAI'91)*, pages 466–471, 1991.

[128] M. Schmidt-Schauß and G. Smolka. Attributive concept descriptions with complements. *Artificial Intelligence*, 48(1):1–26, 1991.

[129] M. Shanahan. *Solving the Frame Problem: A Mathematical Investigation of the Common Law of Inertia*. The MIT Press, 1997.

[130] E. Sirin and B. Parsia. Pellet system description. In *Proc. of the 19th Int. Workshop on Description Logic (DL 2006)*, volume 189 of *CEUR Electronic Workshop Proceedings,* `http://ceur-ws.org/`, 2006.

[131] E. Sirin, B. Parsia, B. Cuenca Grau, A. Kalyanpur, and Y. Katz. Pellet: a practical OWL-DL reasoner. Technical report, University of Maryland Institute for Advanced Computer Studies (UMIACS), 2005.

[132] M. Spielmann. Verification of relational transducers for electronic commerce. *JCSS.*, 66(1):40–65, 2003. Extended abstract in PODS 2000.

[133] C. Stirling. *Modal and Temporal Properties of Processes*. Springer, 2001.

[134] J. K. Strosnider, P. Nandi, S. Kumaran, S. Ghosh, and A. Arsanjani. Model-driven Synthesis of SOA Solutions. *IBM Systems Journal*, 47(3), 2008.

[135] J. D. Ullman. Information integration using logical views. In *Proc. of the 6th Int. Conf. on Database Theory (ICDT'97)*, volume 1186 of *Lecture Notes in Computer Science*, pages 19–40. Springer, 1997.

[136] J. D. Ullman. Information integration using logical views. *Theoretical Computer Science*, 239(2):189–210, 2000.

[137] R. van der Meyden. Logical approaches to incomplete information. In J. Chomicki and G. Saake, editors, *Logics for Databases and Information Systems*, pages 307–356. Kluwer Academic Publishers, 1998.

[138] M. Y. Vardi. The complexity of relational query languages. In *Proc. of the 14th ACM SIGACT Symp. on Theory of Computing (STOC'82)*, pages 137–146, 1982.

[139] M. Y. Vardi. An automata-theoretic approach to linear temporal logic. In F. Moller and G. Birtwistle, editors, *Logics for Concurrency: Structure versus Automata*, volume 1043 of *Lecture Notes in Computer Science*, pages 238–266. Springer, 1996.

[140] M. Y. Vardi. Reasoning about the past with two-way automata. In *Proc. of the 25th Int. Coll. on Automata, Languages and Programming (ICALP'98)*, volume 1443 of *Lecture Notes in Computer Science*, pages 628–641. Springer, 1998.

[141] M. Y. Vardi and P. Wolper. Automata-theoretic techniques for modal logics of programs. *J. of Computer and System Sciences*, 32:183–221, 1986.

[142] J. Vergo. Artifact-based transformation of IBM Global Financing, a case study. In *Proc. of 7th Int. Conference on Business Process Management (BPM 2009)*, 2009.

[143] X. Zhao, J. Su, H. Yang, and Z. Qiu. Enforcing constraints on life cycles of business artifacts. In *TASE*, 2009.

[144] G. Zhou, R. Hull, R. King, and J.-C. Franchitti. Using object matching and materialization to integrate heterogeneous databases. In *Proc. of the 3rd Int. Conf. on Cooperative Information Systems (CoopIS'95)*, pages 4–18, 1995.