**Terminologies**

**Definition 1.** A **concept** is a relation. If \( D_1, D_2, \ldots, D_n \) is a list of domains, then \( X \subseteq D_1 \times \ldots \times D_n \) is a concept. Each domain \( D_i \) is finite and unknown a priori.

**Definition 2.** If \( X \) is a concept, the **structure of a concept**, denoted by \( X.atts \), is described by the list of domains (attributes). The \( i \)-th element of \( X.atts \) is represented by \( X.atts[i] \); its name by \( X.atts[i].name \) and the associated domain by \( X.atts[i].domain \).

**Definition 3.** (\( X.insts \) Extension of a concept) The enumeration of all instances of a concept.

**Definition 4.** (\( X.i \) Intension of a concept) A description of all instances of a concept.

**Definition 5.** (\( X.D \) Operational definition of a concept) A procedure that specifies how to obtain the set of instances of a concept.

**Definition 6.** (Ground Concept) A concept whose instances are retrievable from data sources using a set of predefined operations.

An **instantiator** is a component that is able to interact with a data source and retrieve instances of a particular concept.

The core of an instantiator is an **iterator**, which is completely specified by the name of the program to be executed and the list of parameters that control the behavior of the iterator.

?? An instantiator is specified by:
?? An iterator
?? An assignment of values for the iterator’s parameters
?? A mapping indicating how to create an instance of a particular concept \( X \) based on the information returned by the iterator
?? The query capabilities (a list of conditions) offered by the data sources when it is accessed thru this instantiator.

If \( X \) is ground concept, then its operational definition is the set of instantiators that can be used to retrieve instances of \( X \).

**Definition 7.** (Compound Concept) Concepts obtained by applying selection, projection, vertical integration or horizontal integration over existing concepts.

**Definition 8.** Two concepts \( X \) and \( Y \) are structurally equivalent if \( |X.atts| = |Y.atts| \) and the attributes are compatible list-wise. Two attributes are compatible if they have the same domain or there is a natural transformation between the two attribute domains.
Given two concepts X and Y are structurally equivalent,

Selection:
\[ Y_D = \varphi_s(X) \] where s is a conjunction of built-in predicates.

Projection:
\[ Y_D = \varphi_p(X) \] where p is a list of functions applied over X attributes.

Vertical Integration:
Let X, Y, Z be three concepts and \( X.atts = |Y.atts| \cdot |Z.atts| \), then
\[ X_D = \varphi_p(\varphi_s(Y \times Z)) \] or in general,
\[ X_D = \varphi_p(\varphi_s(Y_1 \times \ldots \times Y_n)) \]

Horizontal Integration:
Let X, Y, Z be three structurally equivalent concepts, then
\[ X_D = Y \cup Z \] or in general,
\[ X_D = \varphi_p(\varphi_s(Y_1 \cup \ldots \cup Y_n)) \] where union operation is over bags.

General form of the operational definition of a compound concept:
\[ X_D = \varphi_p(\varphi_s(Y_1 \cdot \ldots \cdot Y_n)) \]
where \( Y_i \) is a predefined concept and the operator “\( \cdot \)” is one of the compositional operations defined in INDUS such as \( \lor \) or \( \times \).

In INDUS, a global ontology consists of the set of concepts that are used to describe entities and relationships in the domain of discourse.

Queries are expressed in terms of concepts in the global ontology.

The global ontology can be extended by defining new concepts in terms of existing concepts using a well-defined set of compositional operations.

Global ontology is used to hide the complexity of accessing and retrieving information from data sources.

Semantics of user-defined concepts in the global ontology are mapped to the semantics associated with the ground concepts (i.e., concepts in the local ontologies) using compositional operations and/or predefined or user-supplied functions.

**Structure of INDUS**

**Physical Layer:**
- Communicate with the information sources
- Federated database architecture

**Ontological Layer:**
- Contain global ontology specified by users
- Data-source specific ontologies (i.e., ground concepts)
- Mappings to local ontologies associated with the information sources
Transformation of queries into execution plans

User Interface Layer:
- Define ontologies
- Post queries and receive answers

Implementation of INDUS

There are five modules of the data integration component of INDUS:
- **GUI**
  - Interact with INDUS
  - Describe user ontologies
  - Define operational definitions of ground concepts, compound concepts, and queries
  - Register iterators
  - Execute queries
- **Common global ontology area**
  - Manage the repository where definitions of ground concepts, compound concepts, queries, and iterator signatures are stored.
- **Instantiator library**
  - Java classes of iterators and instantiators
  - Maintain clear separation of users’ ontologies and instantiators
- **Query resolution module**
  - Convert queries expressed by global ontologies into queries for relevant data sources
- **Private user workspace**
  - Store answers for posted queries or partial or temporary results as relations tables
  - Store materialized views or tables of query expression trees

Four different user roles of INDUS

- **As a domain scientist:**
  - define ontologies, compound concepts, and queries
  - have knowledge of the relevant domain, some familiarity with the data sources and their capabilities
- **As an ontology engineer:**
  - expand iterator library, define ground concepts for new data sources
- **As an administrator:**
  - install INDUS software, set up and manage databases
  - user management
- **As a developer:**
  - add new compositional operations
  - modifying GUI and query resolution module