Status and Design Rationale of KAON Language

This document explains the features implemented in KAON until October 2002. Further, it discusses the design rationale and explains why certain features were selected.

Basic Design Goals

The target application domain of KAON is e-business on the Web, which as a consequence stressed the following requirements:

1. The system must be scalable. Our goal is to handle ontologies on the order of magnitude of 100,000 concepts and 1,000,000 instances.
2. The system should enable concurrent access by many users (a critical requirement on the Web).
3. The system should be based on well-known technologies. Companies are typically skeptical of basing their business processes on technologies that don't have wide acceptance.
4. The system should be easy to use and implement.

Some examples of applications that can be realized using KAON are:

1. Generation of ontology-based Web portals,
2. Ontology-enabled e-business, possibly with integration with development in Web services (e.g. enriching product catalogs using ontologies),
3. Ontology-based knowledge management (e.g. enriching traditional document management systems with ontologies to provide ontology search),
4. Ontology-based information integration.

Most technology-related requirements may be fulfilled by selecting relational databases as the target persistence medium. Databases scale well to huge data sets, provide excellent concurrency capabilities and are widely available.

KAON Ontology Language

KAON language contains most RDFS primitives, but with cleaned-up semantics. Here is a summary of its features:

- Model and meta-model are cleanly separated. E.g., subClassOf, domain etc. are modeling primitives and are not accessible as entities in the model.
- Usual RDFS modeling constructs are supported – concepts, subconcepts, properties, subpropertes, instances and property instances. The concept and property hierarchies may not contain cycles. Instances may be instances of multiple concepts. Data types are currently not supported.
- Symmetric, transitive and inverse properties are supported and KAON can use them in inference procedures.
- Properties have domains and ranges, which are sets of concepts. Additionally, for each (property, domain concept) pair, a minimum and maximum cardinality may be determined. The semantics of this is discussed later in this document.

- Meta-class modeling is supported. Ontology entities are not strictly separated into concepts and instances. When a concept is created in the model, concept's spanning instance is created as well. The semantics of this is discussed later in this document.
- Ontology modularization is supported. Information is organized into OI-models (ontology-instance models), which may contain classes, properties and instances. OI-models can include other OI-models. Although the including model contains all information from source models, each entity retains information about where it comes from, thus allowing evolution of modularized ontologies. OI-models provide the boundary for the closed-world assumption.
- Lexical information is explicitly present in the model. Labels, synonyms etc. are represented as instances of Label, Synonym etc. concepts, which are assigned to spanning instances of appropriate ontology entities. In such way ontology can easily be queried for concepts according to their labels.

Implementing KAON Using Relational Databases

KAON language has been implemented within Engineering Server. The name stems from the fact that this type of the server supports ontology engineering, which introduces significant limitations on the organization of the database schema. The schema is presented in Figure 1.

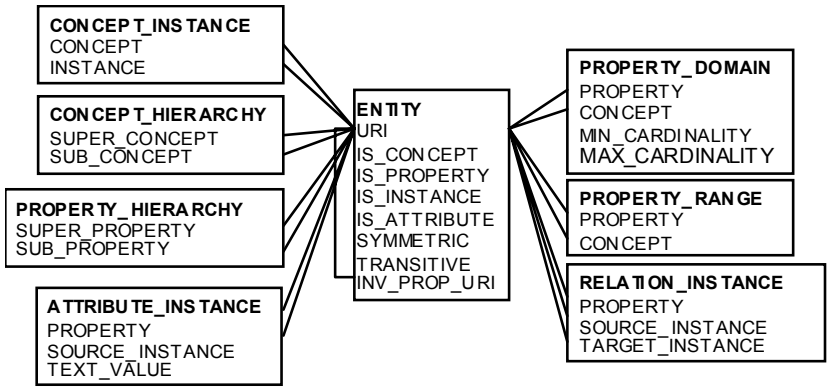

Figure 1. Engineering Server Database Schema

An alternative schema would be to store each concept in a separate table. However, the schema has to support ontology engineering, during which creation and deletion of concepts are frequent operations. Storing concepts in separate tables would result in concept creation and deletion not being transactional operations.

An implementation of KAON for mapping any relational database schema to an ontology is in the works.

Semantics of Domains, Ranges and Cardinalities

In KAON it is possible to say that a person can have at most one name, or that a computer can contain at most five expansion cards. We consider this very important in case of mapping to relational databases. A field with cardinality between 0 and 1 may be mapped to a single nullable field. Cardinality between 0 and infinity signals that a mapping through a foreign key must be made.

KAON treats domains, ranges and cardinalities as constraints. For example, if property *name* has concept *Person* as the domain, then if instance p1 is not an instance of the concept Person, it cannot have the *name* property, as this validates the domain constraints.

An alternative interpretation would be to allow p1 to have the property *name* regardless of the concepts to which p1 explicitly belongs. Then, from the domain constraint of the property *name* one can conclude that p1 must be a *Person*. KAON does not support this. Similarly, KAON will not use cardinalities for classification.

This approach has been chosen based on our observation how people usually use object-oriented models. Usually a conceptual schema of the domain is first built. The schema is then populated with instances. Finally, instances are connected using property instances. Thus the question "Can I apply this property to this instance?" is more important than "I applied these properties; to which concepts does this instance belong?" Alternatively, users will ask "To connect i1 and i2 using p, of which concepts must i1 and i2 be?", rather than "I connected i1 and i2 using p; to which concepts do i1 and i2 now belong?". The schema is seen as a guide on how to populate the instances, not the other way around.

These usual questions can also be efficiently implemented in relational databases.

Semantics of Meta-modeling

In KAON concepts have the usual interpretation of sets. However, we observed that in traditional object-oriented models it is often unclear whether something is a set or a set member. Actually, the choice depends on the point of view. When talking about apes as a species, an *Ape* will be an instance of the concept *Species*. However, when talking about a particular ape1, it will be treated as a member in the *Ape* set. Often it is necessary to talk about the same entity from the different perspective within the same ontology.

Therefore, each concept in KAON has a spanning instance with the same URI. E.g., an ontology may contain statements about the *Ape* as a concept and about *Ape* as an instance.

Currently, the semantics of KAON strictly separates the statements about the spanning instance from the statements about the concept's instances. For example, the statement that *Ape* as instance of concept *Species* has *habitat Africa*, doesn't say anything about ape1. In deed, ape1 may well live somewhere else. Statement about *Ape* instance is nothing more than that – just another statement about an instance.

In future we intend to extend these features and to provide more semantics between statements about the spanning instance and statements about instances of the concept (e.g. default values). We will integrate this into our KAON query language.

Currently meta-modeling allows simple tagging of concepts with lexical information. A concept's label is represented as an instance of the *Label* concept, which is attached to the spanning instance of the concept. In such way the manipulation of lexical information follows the same principles for manipulation of other ontology information.

Closed-world Assumption

OI-models provide a natural boundary for the closed world assumption. All relational databases are based on the closed-world assumption. If a fact is not present in the database, it will not be returned as the result of the query.

From our experience, most business applications have a natural closed world assumption boundary. Document management systems contain a limited set of documents and on-line catalogs contain a final set of products.

Unique Names Assumption

In KAON we employ the unique names assumption, thus greatly simplifying the implementation of the system. For example, enumerating instances of a concept can be simply done by the following SELECT statement:

```
SELECT in.URI FROM Entity in, Entity co, ConceptInstance ci
     WHERE co.URI='Person'
          AND ci.concept=co.uri
```

In case the model contained information that the *Person* is equivalent to *Human*, then we'd have to make a union.

Things are even more complicated if we remove the UNA for instances. Then for each instance there should be some processing for determining if the instance is the same as some other instance, which would significantly degrade the performance.

Removal of UNA introduces significant complications to the ontology API – subconcepts of a concept should be returned as a set of sets of equivalent concepts. Cycles in the hierarchy should be allowed, which results in more difficult ontology evolution.

In order to simplify the implementation of the core language, we therefore employ the unique names assumption.

Alternatives to UNA

It is unrealistic to require that in the Web semantically same entities will always have the same URI. For example, instances "Alexander Maedche" and "A. Maedche" denote semantically the same person, but have different URIs. In that case it seems reasonable to be able to say that these two entities are equivalent.

Even in such seemingly simple case, we argue that in realistic information integration scenario instance equivalence doesn't solve much of the problem. Assume that two departments in a company want to integrate their databases about people, where each database contains thousands of entries. Establishing explicit equivalence relations between such large data sets manually will be difficult, if not impossible. Rather, the integration should be done by the rule saying that the names of people should be matched by reducing the name form the first database to the initial. In such way a large percentage of the database will be integrated automatically – only the ambiguous cases will need to be solved manually.

Most realistic information integration scenarios are even more complex, requiring joins and filtering of datasets. E.g., a database containing concept *Person* may need to be integrated with a database contain concept *Employee*. The integration should be performed by taking a subset of people that are known to work in the company or any of its subsidiaries.

Finally, the most difficult scenario is when new objects have to be generated. For example, a database may contain concept *Document* with a property *authorName* and the goal is to automatically populate the concept *Author* with instances based on the unique values of the *authorName* property. This approach typically involves the application of Skolem functions for generation of URIs of new instances.

To summarize, the problem of reconciling ontologies on the Web is multi-faceted. From our experience equivalences between ontology elements alone don't solve much of the problem – more powerful mapping mechanisms are needed. Hence, we opted to keep the basic language small and easy to implement. On top of that simple basis we plan to introduce a more sophisticated mapping layer that will include equivalences, but will provide also other important features.

Open Issues

The biggest open issue is querying KAON data. The query language should provide support for querying schema as well as instances, means to restructure data and to generate new objects. Further, the language should provide support for evaluating simple non-recursive rules, such as ‘grandparent is a parent of a parent’. The query language will be integrated with meta-class modeling, allowing shifting the focus of the query from the instance to the concept level and view versa.

Data types should be supported.