ENHANCEMENT OF EXPERT SYSTEMS WITH OBJECT-ORIENTED DATABASE FEATURES

G. Al-Shorbagy*, M. Zaki, A. Rafea***, F. Essa**, Mahmoud Rafea***

*** Central Laboratory for Agriculture Expert System (CLAES)**

**** Computer Department, faculty of engineering, Azhar University**

***** Faulty of information and Computers, Cairo University**

Abstract

Intelligent systems are computer-based systems that use knowledge and reasoning techniques to solve problems. But they haven't the ability to manage a large amount of data like a DBMS's. Many expert system applications have large amount of data, thus we need to couple both the knowledge base and database systems. In this work we built two systems named Loosely Coupled System (LCS) and Tightly Coupled System (TCS).

LCS statically couples the knowedge base and the database. This means that LCS must first invoke all the required data from the database. Secondly, it stores them in the working memory of the system. Finally, it starts the interactive session. TCS dynamically couples the knowledge base and database. This means that the interactive session starts first. Whenever the system needs data from the database, the inference engine asks the database to afford the specified data.

Both LCS and TCS developed to couple KROL knowledge base with the external storge utility of the SICStus object prolog languag. An Object-Oriented Layer (OOL) has been built on the top of the external storage utility of SICStus Prolog to handle the database objects and to convert them to the external storage format. LCS and TCS are tested by building many applications in the agriculture domain, and their performance is investigated.

1. Introduction

The integration of a knowledge-based system and a database system is important to a large class of users and applications. Such integration can be achieved in two ways that are loosely and tightly coupling [Heng 97]. Moreover, the integration, as such, must be simple, accurate and maintainable. These goals ware the aims of different researchers and systems such as the following.

- [Claude 88] is composed of inference rules written in PROLOG extension language (G-LOGIS) and set of facts managed by a DBMS (G-BASE). G-LOGIS is an object-oriented implementation of PROLOG, while G-BASE is an object-oriented DBMS that manages the object structured fact base. G-LOGIS and G-BASE are able to work in harmony because they are both based on object-oriented paradigm.

- The tight coupling of PROTUS shell and the ORION object-oriented database system is introduced in [Nat 88]. A PROTUS/ORION integration is developed in the Advanced Computer Program (ACM). The addition of the database management system to PROTUS has simplified the tasks of knowledge engineering. The results suggest that further experimentation with such a system is worthwhile.
- A Dictionary Interface for Expert Systems and Databases (DIFEAD) is introduced in [Al-zobaidie 88]. DIFEAD contains a kernel called Metalevel Component (MLC). Through MLC the ES can be accessed. There are three main modules in the MLC: UIM (User Interface Module), MQM (Metadata Query Module) and DUM (Data Update Module). The UIM interacts with the user. The MQM finds out whether some data item can be found in the application or not. If no answer can be found in the application, the MQM informs the UIM to request the missing information directly from the user.
- A logic-programming environment (EPSILON) that allows the use of data stored in a relational database from a logic program is introduced in [Selmin 90]. EPSILON is a prototype developed in the context of the European ESPRIT project. It is built on the top of commercial PROLOG (BIMProlog) and DBMS (Informix), running on standard Unix environment.
- A design and implementation of a database interface for the LIFE system are introduced in [Marcel 92]. LIFE is logic programming, such as Prolog, but it is extended with an elegant type system supporting a facility for structured type inheritance. Also, it is integrated with a functional language and allows the interleaving of relational and functional expression. Together, these features allow powerful high-level expressions and complex constraints on data-objects.
- An integration strategy of an expert system in oncology therapy with a clinical history database and a user interface adapted to the needs of the medical environment are proposed in [Maria 96]. The objective of this system is to design a system that is easy to use and provides all functions needed by the medical staff. The final system has consists in first integration three developing tools and then using them in order to construct the complete application. The result of this work is a global tool for the development of clinical information systems with embedded intelligence module.
- A new loose coupling approach (simple coupler) based on predefined SQL was introduced in [Heng 97]. It is an integrated computer system to couple an existing DB system with ES through a coupling module in a multi-tasking environment. A multi-tasking platform, e.g. UNIX at

workstations or MS-Windows at personal computers, is necessary because there are actually three programs (coupling module, DB application and ES) running in the memory. The communication between the coupling module and both the ES and the DB is through some channels, e.g. UNIX pipes, or MS-Windows DDE [Dynamic Data Exchange (DDE) or Object Linking & Embedded (OLE)].

This work introduces two coupled intelligent systems which are: TCS and LCS. The two systems have been built at Central Laboratory for Agriculture Expert System (CLAES), Egypt.

Actually, CLAES needs an object-oriented database management system that can be coupled with the expert system that is developed there. This integration necessitates some modifications of the inference class. Therefore, we introduce this research to build an object-oriented layer on the top of the external storage utility of the SICStus prolog and we provide a couple between it and the KROL inference engine.

KROL is a **K**nowledge **R**epresentation and **O**bject **L**anguage that combines logic, object-oriented and rule-based programming paradigms [Shaalan 98]. The KROL language is an extension to SICStus Prolog Objects language [SICS 1995]. The main facilities provided by KROL are the expressive power to represent complex concepts. Also, from the KROL facilitates, the integration of many problem solvers and knowledge representation schemas in one application.

Integration of ES with DBMS constitutes the development of a single system that handles all data by itself, if necessary in a dedicated database subsystem. Any programming system is in fact an integrated system, in that all data is handled by its internal data handling facilities, so forming a rudimentary database. This is called *elementary database management [Marcel 92]*. There are some serious drawbacks to this method:

- The internal data handling facilities cannot handle large amounts of data, since these do not fit in main memory. Even if the data would fit in memory, evaluation of the data will not be efficient, because the expert system generally does not offer any facilities to optimize data handling, such as indexing.
- Most expert systems offer only very elementary data management facilities. There are no data dictionaries and no generalized set-oriented operations. There are no facilities to ensure the persistence and integrity of the stored data such as locking, atomically of transactions and concurrency control

These problems are overcome by integration of expert system and database, where data handling is performed by the database subsystem, which uses external files to store data and efficient

indexing schemes to retrieve it. Figure 1 shows two architectures for integration approach. Architecture ‘a’ shows an expert system (ES) with a database subsystem (DBMS).

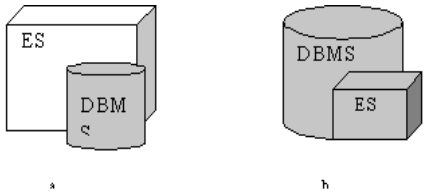

Figure 1 Architectures for the integration approach

This subsystem handles data at the request of the ES system, and is therefore well suited for extension of an expert system with database facilities, or for a database system in which we want to use a logic programming language as database language. Architecture ‘b’ is a database system that includes an expert system (ES), implemented in a logic language, containing the optimization component. This architecture is well suited for enhancement of a database with an optimization component.

In a coupling ES with DBMS, an interface between two independent systems must be developed. The two systems are logic programming system and a database system. Both systems preserve their individuality; an interface between them provides the procedures required for bringing data from the persistent database system into the logic-programming environment and vice versa. There are two types of coupling: Loose and Tightly couplings.

- Loose (static) coupling. The interaction between the programming system and the database takes place independently of the actual computation process in the programming system. Typically, coupling is performed at compile time (or at program load time, with interpreters) by extracting from the database all the facts the program might require. After the data has been loaded in the programming system, it is stored in the internal database and handled by techniques as integration technique. Hence a system can employ both a coupling with a database system and integration, since the programming system can have its own database subsystem.

- Tight (dynamic) coupling, the computation process, for example the inference process in a logic programming system, drives the interaction with the database system by extracting the specific facts required to answer the current goal or sub goal at the time they are needed. Thus, coupling is performed whenever the logic programming system needs more data from the database system to proceed with its inference. The important design issues are:

- Should all data be stored in the internal database or retrieved at every step of the inference process again?

- Do the system only load data that is actually needed, or do we anticipate on the data demand, and also load data from which we expect that it will be needed soon?

Figure 2 shows architectures of loose and tight coupling. The interaction consists of ***retrieval:*** the Expert System (ES) requests data from the database system, which replies by sending the requested data, and ***updates:*** the expert system sends data to the DBMS that have to be inserted or deleted.

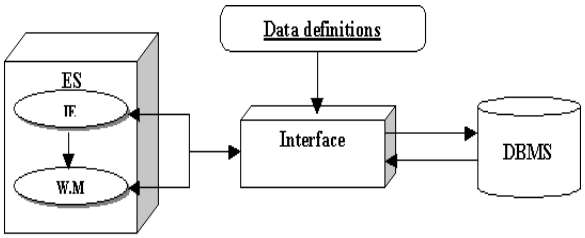

Figure 2 The coupling architecture

As Heng-Li Yang [Heng 97] states, there are four ways that expert systems can access databases.

1. Elementary data management *within the Es;*
2. Generalized data management within the ES;
3. Loose coupling of the ES with an existing DBMS; and
4. Tight coupling of the ES with, an existing DBMS.

One should note that, we used "loose coupling" to mean *static* coupling, i.e. data extractions occur statically before the actual operation of the ES; and "tight coupling" to mean *dynamic* coupling, i.e., during the same ES session, many different portions of the external database may be required at different times.

2. LCS and TCS Systems Design

The main idea of this work is the coupling of the expert system with the database. The database that is used in this work is an object-oriented database. The coupling of ES and database would be achieved using two systems. The first system entails building a Loosely Coupled System (LCS) and the second entails building a Tightly Coupled System (TCS). The Object oriented database system consists of two components. The first is the SICStus prolog external storage utility. The second is an object-oriented layer on the top of the SICStus prolog external storage utility. The object-oriented layer handles the database objects and converts them to the external storage format.

In the following, the design of the object-oriented layer and the complete design of the two systems will be discussed in details. In addition, a comparison between the two systems will be discussed.

2.1 The Loosely Coupled System Design

The loosely coupled system is shown in figure 3. In this system, the communication between ES and DB is achieved through an interface program that maps the physical database with the attributes in working memory. In this system, all values, which are needed from the database, are collected once. The system supports the following scenario:

- Determine the expert system values that must be stored in database.
- Get these values from the database.
- Mapping these values with expert system attributes.
- Store these values in working memory.
- Start the expert system session.

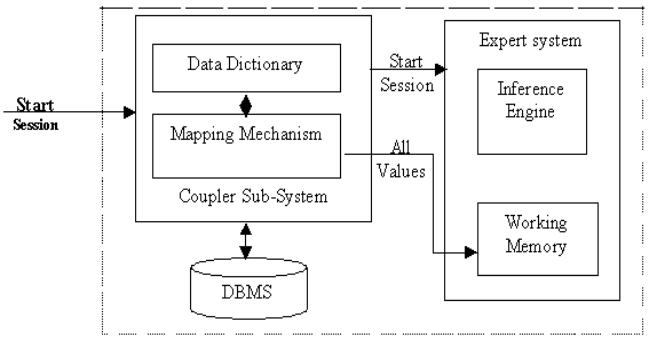

Figure 3 Loose Coupling Architecture

To satisfy this scenario, the system must include the following components:

1. Data dictionary that includes all expert system database attributes. The dictionary contains the field name of the database and the corresponding attribute name of the expert system.
2. Mapping mechanism that collects all the database fields and stores them in the working memory.

The design procedure of LCS is pointed out in the following:-

Input : Case_Id <database key that identifies the needed data>

Procedure:

1. The Mapping Mechanism would determine from the Data Dictionary all the needed fields and there relations (Database Table).
2. For each Field-Relation pair do
   - i. Use Case_Id to fetch the value for each field from the relation.
   - ii. Determine the corresponding Object-Attribute pair from the Data Dictionary.
   - iii. Assert the field value into the Working Memory to the corresponding object-attribute.
3. Call the Inference Engine to start the ES Session.

2.2 The Tightly Coupled System Design

The tightly coupled system is shown in figure 4. In this system, the communication between ES and DB is achieved through a coupling mechanism that links the ES attributes with database fields. Whenever the ES needs a database field value and this value does not exist in the working memory, the ES uses the coupler sub-system to retrieve this value from the database.

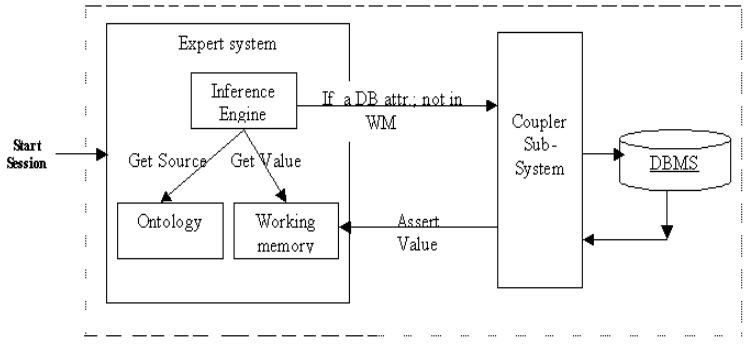

Figure 4 Tightly Coupling Architecture

During ES session, if the inference engine reasons about database attribute, then the following scenario would be occurred:

- Determine the database object that includes the required attribute.
- Determine the name of database field that corresponding to the required attribute.
- Send to the coupler sub-system to query OO Layer to retrieve the field value from the database.
- Save this value in the working memory.
- Complete the session.

To satisfy this scenario, the system must include the following components:

Coupler sub-system that has the ability to open a channel between the DB and ES whenever ES needs values from DB. The sub-system includes an internal method that queries OO Layer to retrieve the field value from the database. In the TCS system, the inference engine has its ability to determine the source of value for each attribute. Also, it has a special mechanism for each source. According to this mechanism, the inference engine accesses the source to get the attribute value. There are different sources of values and the database is one of these sources. The default source of attribute value usually is the user. If the system failed in getting the attribute value from the given source, the system would ask the user about this value. This makes the system more powerful.

The design procedure of LCS is pointed out in the following:-

Input : Case_Id <database key that identifies the needed data>

Procedure:

1. Start ES Sessiuon, whenever the Inference Engine reasons about any attribut do the following
    i. The Inference Engine searches in working memory for this attribute.
    ii. If this value is found then Take it and stop.
    iii. Else, Get the source of value of this attribute from the Ontology.
    iv. In case of source of value is database do the following:
        1. Determine the Table and the Field names from the ontology for this attribute.
        2. Call the Coupler Subsystem to Fetch this field from the database according to the Case_Id
        3. The Coupler Subsystem would Assert the field value into the Working Memory to the corresponding object-attribute.

2.3 The Object-Oriented Layer (OOL)

The SICStus prolog external storage utility handles storage and retrieval of terms on files. By using indexing, the store/retrieve operations are efficient [SICS 95]. The object-oriented layer (OOL) supports the main features of database and object oriented technology. The supported database features are constraint checking, query, views and many supporting methods that allow the user to handle own databases. The supported object-oriented features are inheritance, reusability and handling of the complex object. Also, OOL handles the database objects and converts them to the external storage format.

The proposed object-oriented layer and the interactions with its environment are shown in figure 5. The object-oriented layer contains three main objects. They are: 1) DBMS object, 2)-view object and 3) access-constraint object. Each object has its own methods. There are three different users that can access the database. The first one is the system administrator who has access capability to all objects and methods of the OOL. The second is the database user who can access the database through the application views. The third is another application such as expert system application. The ES can access the database through the DBMS object.

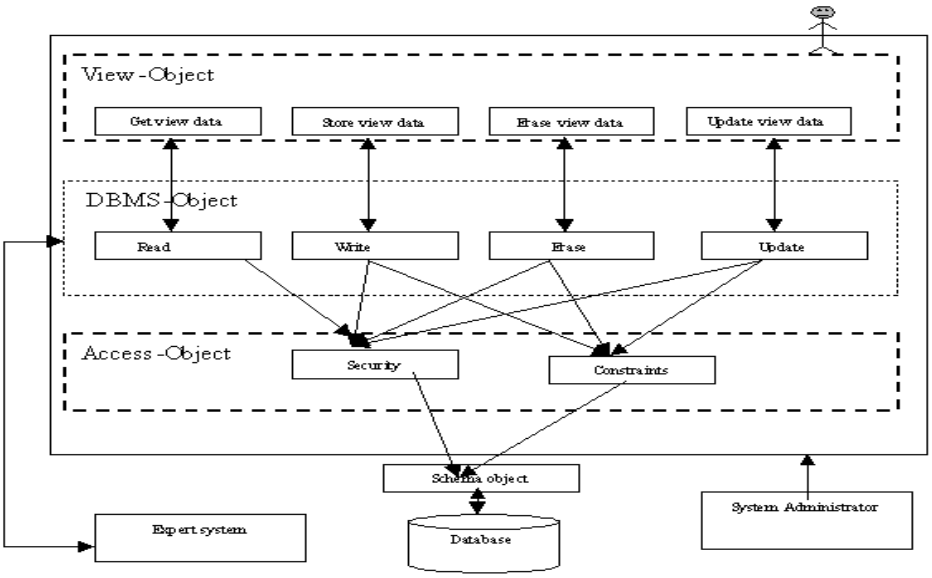

Fig 5 Object-Oriented Layer Design

2.3.1 Data Definition Language (DDL)

The DDL supports all database definitions, which are data items, security, constraints and views. All these definitions are encapsulated within an object in the hierarchy representation of the object scheme. The schema is an important part of any database. So any DBMS provide a general structure to describe schema. This system uses general methods that can be used in schema declarations. The following eight methods give a full declarations of any database application schema.

The definition of database owners

The object-oriented layer classifies the users of the database into two types; owners or others. The owners of any database must be declared explicitly in the schema declaration. More than one owner may have the same database. Therefore, the DDL language contains a method called owner/1. This method supports the security of the database.

The definition of the database relations

This is the most important method of a schema declaration. Through this method, the object-oriented layer can create and open a physical relation. This definition includes the name of a relation on the physical disk, the list of attribute names - this attribute may have zero or more arity-, and the authority of non-owner users to access relation.

Indexing

Indexing in a database is very important to decrease the search time. The DDL language includes a definition of the indexed fields.

View handling

The view handling includes a view name that is the name of a new logical relation created in memory. It also includes the relations' names and fields names that are used in the view. Through this view the user can handle the physical data of the original relations.

Constraint satisfaction

The DDL contains a **constraint method** for each DB operation. When the operation is needed, the OOL first sends a message to the schema object to get the constraint method of this operation. If this constraint is satisfied, the operation is done, otherwise the operation would be canceled and it gives an error message to the user. The problem now is how to define a constraint for each method. The DDL contains three methods. In the first method, the constraints of write to database would be defined. This method is called ***store rules***. In the second method, the constraints of erase from database would be defined. This method is called ***erase rules***. In the third method, the constraints of update database would be defined. This method is called ***update rules***. The language only knows the names of the constraint methods, but the contents of these methods are left to the developer.

2.3.2 Data Manipulation Language (DML)

The DML contains a number of methods that support all events in the database operations. In the following sections we describe each method.

A. DB manipulation methods

A.1 Read from the database method

Input: relation name and key values.

Objective: reads the values from the database relation, which match the key.

Algorithm:

1. Get the user of the system.
2. If the user is the database owner, go to step 6.
3. Check the non-owner authority.
4. If the system gives the read authority of this relation for non-owner, go to step 6.
5. Announce fail.
6. Do the read operation.

A.2 Write to database method

Input: relation name and values.

Objective: writes these values into the database relation if they satisfied the store rules of this relation.

Algorithm:

1. Get the user of the system.
2. If the user is the database owner, go to step 6.
3. Check the non-owner authority.
4. If the system gives the write authority of this relation for non-owner, go to step 6.
5. Announce fail.
6. Get the store rule of this relation.
7. Applying constraints of the store rule on values.
8. If the constraints are satisfied, then write the data into database relation, otherwise announce fail.

A.3 Erase from database method

Input: relation name and values.

Objective: erases these values from the database relation if they satisfied the erase rules of this relation.

Algorithm:

1. Get the user of the system.
2. If the user is the database owner, go to step 6.
3. Check the non-owner authority.

4. If the system gives the erase authority of this relation for non-owner, go to step 6.
5. Announce fail.
6. Get the erase rule of this relation.
7. Applying constraints of the erase rule values.
8. If the constraints are satisfied, then erase the data from database relation, otherwise announce fail.

A.4 Update from database operation

Input: relation name, old values and new values.

Objective: updates the existing old values by the new values in the database relation if they satisfied the update rules of this relation.

Algorithm:

1. Get the user of the system.
2. If the user is the database owner, go to step 6.
3. Check the non-owner authority.
4. If the system gives the write and erase authorities of this relation for non-owner, go to step 6.
5. Announce fail.
6. Get the erase rule of this relation.
7. Applying constraints of the erase rule on old values.
8. If the constraints are satisfied, then go to step 4, otherwise announce fail.
9. Get the store rule of this relation.
10. Applying constraints of the store rule on new values.
11. If the constraints are satisfied, then update the database, otherwise announce fail.

2.3.3. View methods

2.3.3.1. Retrieve view fields data from the database

Input: view name.

Objective: collects view fields' data from the database relations and put them in the view entries.

Algorithm:

1. collect the fields of the views
2. Determine the relations of these fields.
3. For each relation and its fields, apply the read from database algorithm

2.3.3.2. Write view fields data into the database

Input: view name

Objective: collects view fields' data from the view and puts them in the database relations.

Algorithm:

1. Collect the fields of the views.
2. Determine the relations of these fields.
3. For each relation and its fields apply the write to database algorithm.

2.3.3.3. Erase the view fields data from the database

Input: view name

Objective: collects view fields' data from the view and erases them in the database relations.

Algorithm:

1. collect the fields of the views
2. Determine the relations of these fields.
3. For each relation and its fields, apply the erase from database algorithm

2.3.3.4 Update the database values by the view fields

Input: view name.

Objective: collects view fields' data from the view and updates the database relations by these data.

Algorithm:

1. collect the fields of the views
2. Determine the relations of these fields.
3. For each relation and its fields, apply the update database algorithm

2.3.4. Access-Constraint methods

2.3.4.1 Unique method

Input: relation name and unique keys with values.

Objective: checks the uniqueness of these keys inside that relation.

Algorithm:

1. Get the attributes of these relations.
2. Search for existing of these keys value inside the relation data.
3. If these keys are found inside the relation data, announce fail, otherwise announce successful.

2.3.4.2. Functional dependency method

Input: relation name and values of two fields. The second field value is functional dependent on the first field value.

Objective: checks the functional dependency between the two values.

Algorithm:

1. Search for the existing of the first field value inside the relation data.
2. If this value is not found, announce successful.
3. If this value is found inside the database, retrieve the related second field value from the same record in the database.
4. Compare the retrieved valued with the second input value.
5. If they equal, announce successful, otherwise announce fail.

2.3.4.3. Domain constraints method

Input: relation name and field value.

Objective: checks that this field value satisfying the corresponding attributes requirements defined in the ontology.

Algorithm:

1. Determine the object of the corresponding attribute defined in the ontology.
2. Get the attribute facets that contain the type of the attribute and its possible values.
3. Compare the types of the attribute with the value type. If the value does not match the attribute type, announce fail, otherwise check the value with allowed attribute values
4. If the values satisfying the allowed attribute values, announce successful, otherwise announce fail.

2.4 Inference Engine

The inference engine is the main part of the KROL. The block diagram of a complete expert system, which has been built based on KROL, is depicted in figure 6.

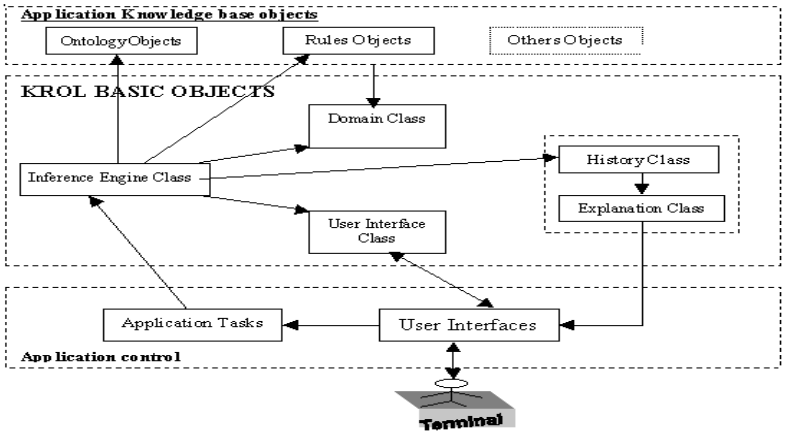

Figure 6 KROL Based Expert System

The system consists of many classes: User Interface, Domain, History and Explanation, and Inference Engine classas.

User interface object "user_jnt"

This object contains methods used by the inference mechanism to get values of attributes whose source of values is defined as user. The methods will differentiate between two cases: the single value attribute and multi-valued attribute.

Inference history object "history"

This object acts as a log object, i.e. it keeps historical information about reasoning. It contains methods used by the inference mechanism. These methods are used to record the action taken during the course of the inference process.

Domain-definitions object "domain_class":

The domain_class object contains the generic methods and the default attribute values of the domain knowledge. This object is the super object for all knowledge base objects in the application.

Explanation object "explanation":

This object contains the explanation methods: why/2 and how/2. The former method is to be activated in response to a user reply "why" during the course of asking the user about an attribute value. The latter method is to be activated in response to a message sent to the "explanation" object in order to query about how an attribute value is concluded.

Inference Engine Class "Inference":

The inference class is the main class of the system. The two coupling systems use the inference engine of KROL [Shaalan 98]. As shown in figure 7, the KROL inference engine has three main reasoninig methods: Methods directly reason about attribute values, Methods directly invoke the inference in order to reason about attribute values, and Methods directly invoke the inference and indirectly reason about attribute values. Each reasoning method calls many functions. The reasoning methods can be explained as follows:

1- Methods directly reason about attribute values. The core method that a developer can use, is get_value/2 method. This method differentiates between two cases during the course of reasoning: the single valued attribute and multi-valued attribute. The method uses the source_of_value/2 meta-attribute to define the value(s) of the given attribute. Consequently, this method only invokes the inference indirectly if the source of value is defined as 'derived'.

2- Methods directly invoke the inference in order to reason about attribute values. The core method that a developer can use, are focus/3 and Rule_head_list/2. The former method, first, collects relations that reason about a particular attribute given by the developer, then fires this rule. The latter method is used to fire a particular rule.

3- Methods directly invoke the inference and indirectly reason about attribute values. There are two defined methods that can be used to deal with rules: conclude/1 and conclude_all/1. The former fires a set of relations defined in a given object. The latter reasons about all the rules defined in a given object and its sibling objects. The inference maintains the open world assumption where the positive or negative values of the attributes are recorded.

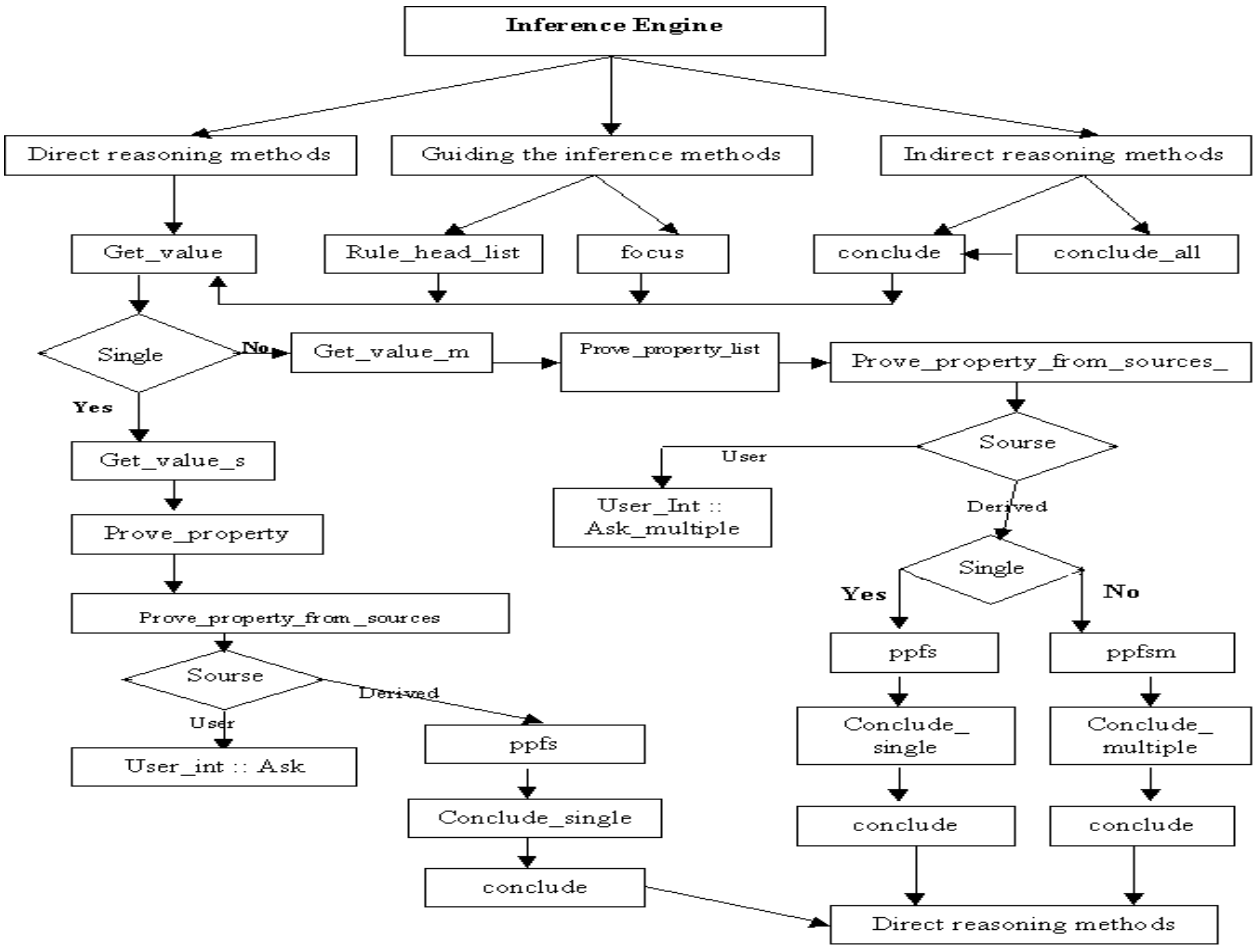

Figure 7 KROL Inference Engine

2.5 Coupling KROL with OOL

The coupling of KROL with OOL provides a powerful expert system that can handle database. We coupled the KROL and OOL to build a loosely coupling system and a tightly coupling system. In the following sections we introduce the two systems and the modifications of KROL.

2.5.1 KROL_based loosely coupled system

In this system, the OOL is added to the KROL objects. Figure 8 illustrates this system. The OOL does not interact with inference engine. Unlike, KROL expert system - the calling of inference is done through the task directly- the task calls the coupler that collects the database and then calls the

inference engine to start session. In the application, the developer must encode the following objects:

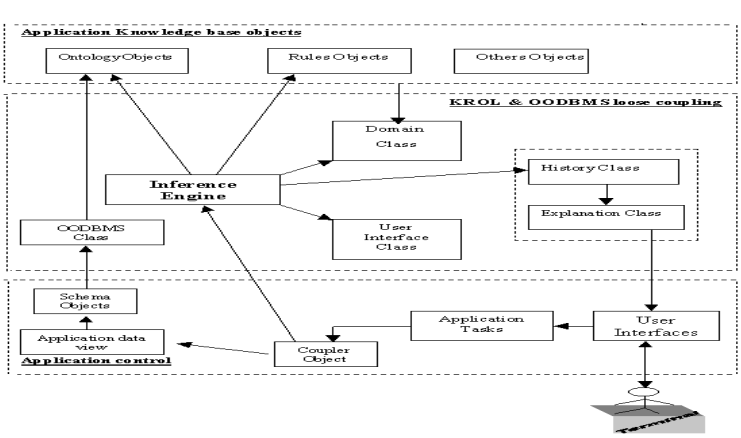

Figure 8 KROL-based Loosely Coupled System

1. Coupler object, which contains the methods that collect all the database fields and store them in the working memory. This is done through the application views and schema objects.
2. Application data view object, which contains the screen form of data views.
3. Schema object: this object contains the database relations and data views definitions

The algorithm of loose coupling is:

```
1)    Determine the session case_id from the farm view object
2)    Call collect_data method with  list of data views.
```

Algorithm of collect_data method is:

```
Input : list of data views [V|Vs]
Algorithm:
        1) If list = [] then exit.
        2) For view V do
                2.1) Determine the view relations and fields using DBMS.
                2.2) Fetch the view relations by case_id.
                2.3) If there are data for this case_id then
                                Assert these data in the view V
                                Get the view V values
                                Call the database_assert with these values and fields.
                        Else
                                Announce Fail
                        Endif
        3) Call colloct data with (Vs)
```

Algorithm of database assert method is:

```
input [Fi,Fs],[Vi,Vs]
        if field list = [] then exit
        Determine for each Fi the coressopnding KB object Oi and KB
        attribute Ai.
        Update the value of Ai in object Oi with value Vi
        Call database_assert with [Fs,Vs]
```

Figure 9 illustrates the interactions between the loosely coupled objects. The interaction starts when the user needs to start expert system session. Since, the task program send a message to the coupler to collect all data from the database and then send theses data to the expert system.

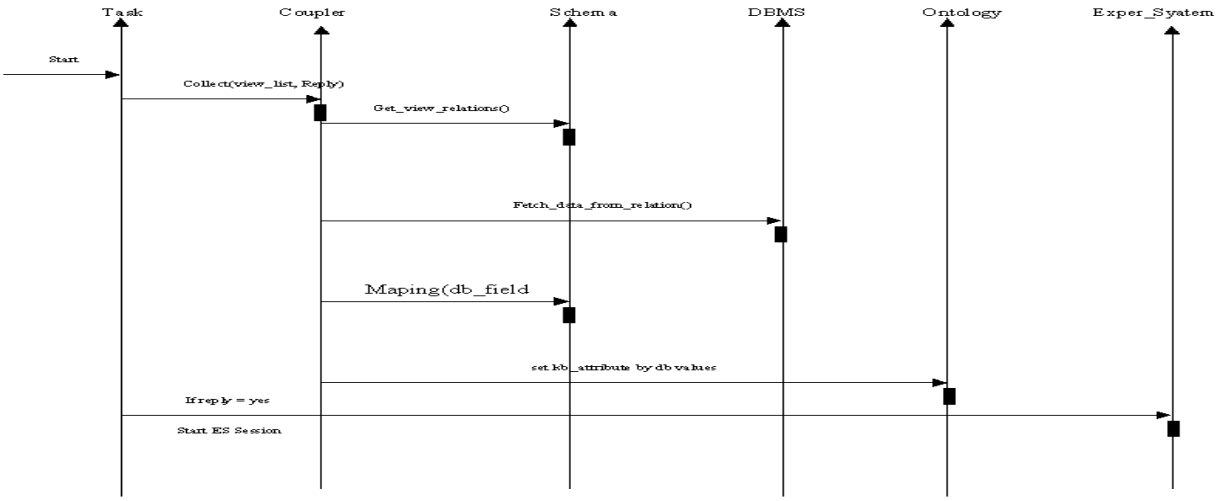

Fig. 9 Interaction Diagram of KROL based Loosely Coupled System

2.5.2 KROL_based tightly coupled system

In this system, the OOL is added to the KROL objects. Figure 10 illustrates this system. The OOL interacts with inference engine. Like KROL expert system, the calling of inference is done through the task directly.

To make the interaction between the inference engine and the OOL, the searching method of the KROL inference engine must be modified to support this situation. The get_value method is updated in inference engine to support database source. Figure 11 shows the modified KROL inference engine.

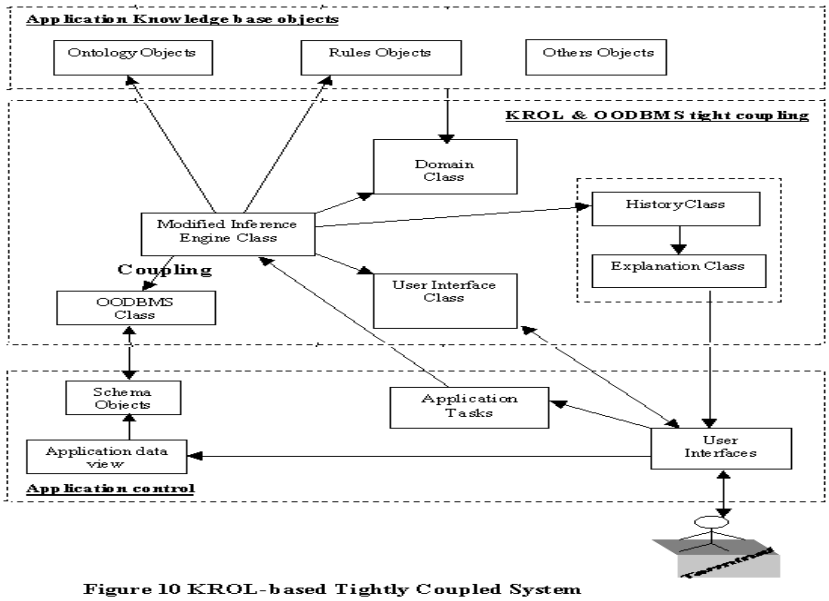

Figure 10 KROL-based Tightly Coupled System

In the application, the developer only encodes the source of database value in the ontology knowledge base.

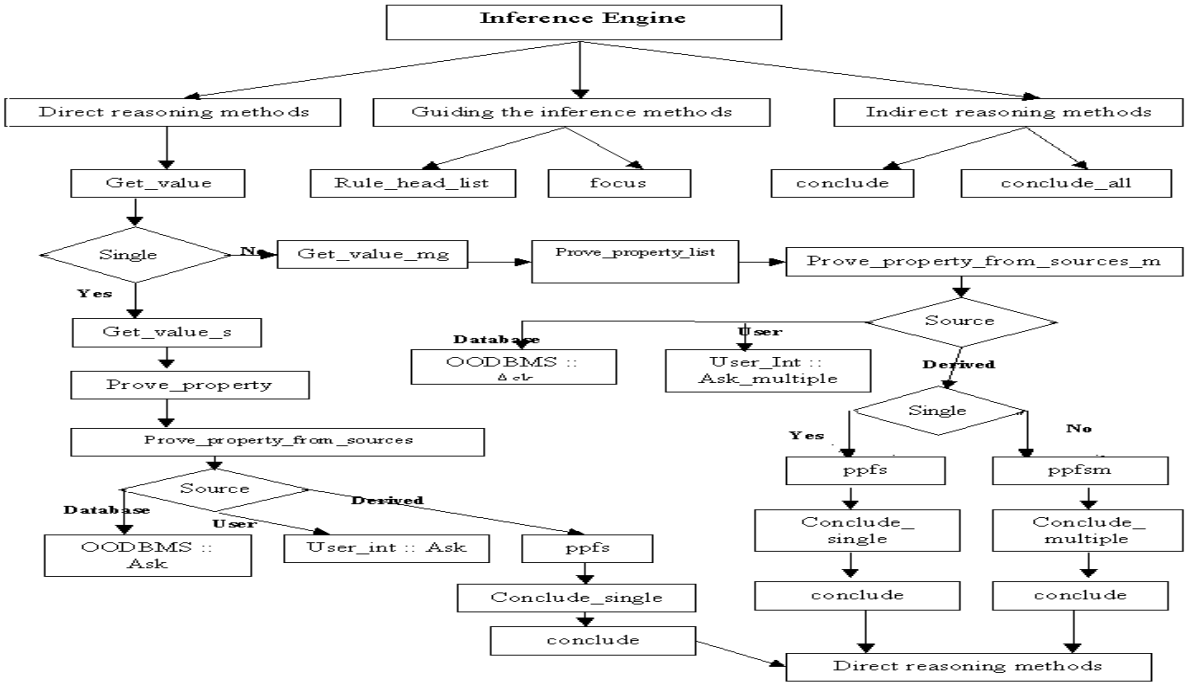

Figure 11 The Modified KROL Inference Engine with tightly coupling

The algorithm of tight coupling is:

1- The inference engine searches in working memory (cash object) for this attribute.
2- If this value is found then
Take it and stop.
Else
Get the source of value of this attribute from the common knowledge base.
3- In case of source of value is database do the following:
3.1 Determine the relation and the field order of this relation.
3.2 Determine the Case ID.
3.3 Fetch this field from the database.
3.4 Assert this data in the knowledge base object.

Figure 12 illustrates the interactions between the tightly coupling objects. The interaction starts when the system needs to get an attribute value.

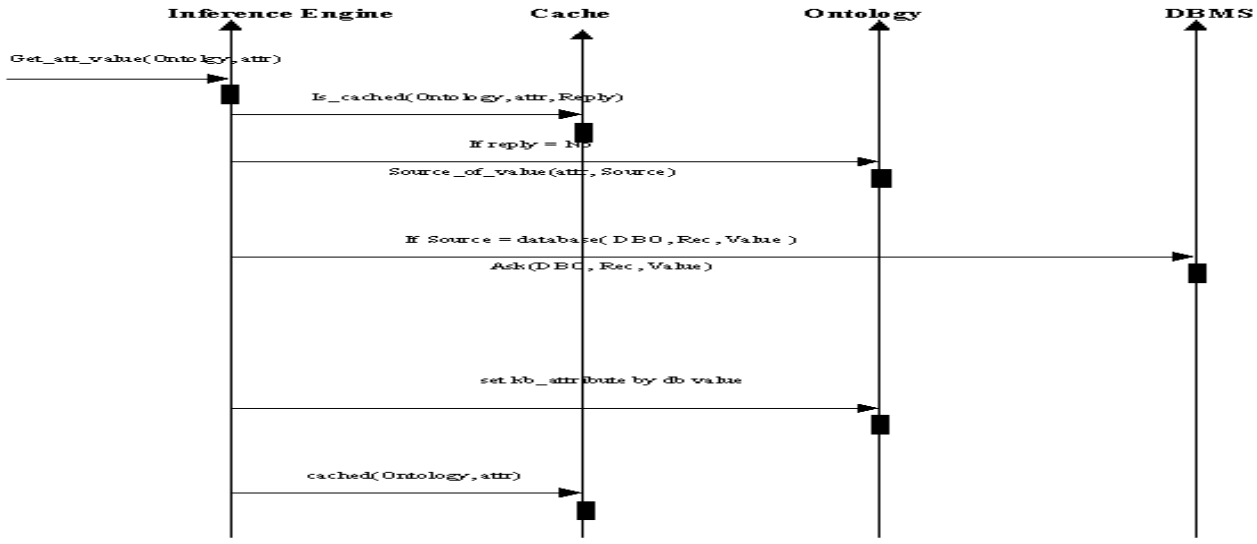

Fig. 12 Interaction Diagram of Tightly Coupled System

3.Case Study

There are three expert systems that are developed - at CLAES - based on the two-coupled systems. These expert systems are CUPTEX, CITEX and TOMATEX. This chapter lists briefly the three expert systems and explains in details the **Cit**rus **ex**pert system (**CITEX**). A comparison between the two approaches is discussed.

- CUPTEX [Dessouki 93] is a complete expert system for managing the cucumber under plastic tunnels from seedling to harvest stages. CUPTEX generates complete irrigation and fertilization schedules, gives the advice for tunnel preparation and provides diagnosis and treatment for total 39 different disorders. CUPTEX was used in many locations. The economic study proved that the net profit of the greenhouses that are cultivated by the expert system is 156.9% compared to greenhouses that are cultivated without using expert system [TR 97].
- CITEX [Salah 93] is a complete expert system for managing the Orange Orchid from seedling to harvest stages. CITEX assesses suitability of Orange Orchid, generates complete irrigation & fertilization schedules and provides diagnosis and treatment for total 38 different disorders.
- TOMATEX is an expert system for diagnosis & treatment for Tomato disorders under low tunnels, greenhouse, and open fields. TOMATEX provides diagnosis and treatment for total 38 different disorders.

3.1 The Citrus expert system (CITEX) [TR 97.5]

There are two implementation versions for coupling CITEX with database which called “Egypt”. The first version is implemented as loosely coupling. The second version is implemented as tightly coupling.

Egypt Database E – R diagram

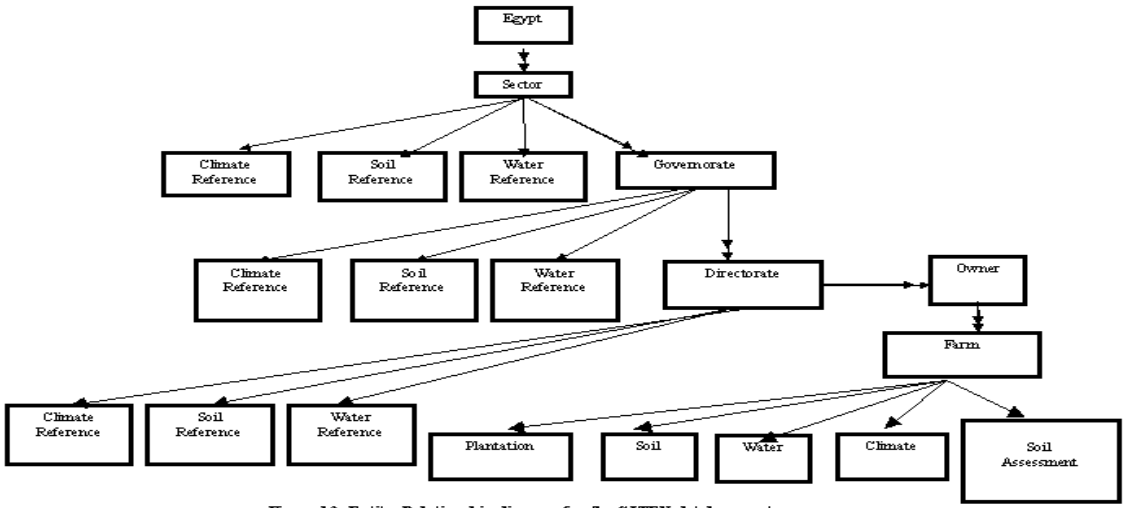

Figure 13: Entity-Relationship diagram for the CITEX database system

figure13 represents the Entity-relationship diagram of “Egypt” database which coupled with CITEX.

The implementation of the loosely coupled CITEX

In the loosely coupling all needed database fields are collected from their view. The collection of the database fields is done through the method collect_data that is a method of object called data_check.

```
data_check :: collect_data([plantation_view, soil_ass_view, soil_view, climate_view] )
```

The parameter of the collect_data is a list that contains view names. These views contain the database fields. In CITEX the views that contain the needed database fields are:

- Plantation view: contains plantation data.
- Soil view: contains soil and water data.
- Soil_ass_view: contains soil data that support the assessment process.
- Climate view: contains climate data for 12 months.

The CITEX developers encode this coupling. The inference engine does not support the database coupling. The database fields are asserted in memory before the CITEX session starts. Also the developers encode the mapping between the knowledge attributes and database fields using attr/2 method.

The implementation of the tightly coupled CITEX

In this version, the CITEX developers update the ontology only. They encode the source of values

```
soil :: {
super(domain_class) &
dynamic texture/1 &
dynamic ec/1 &
dynamic s_status/1 &
source_of_value(texture/1,
[database(farm,soil(S,G,D,O,F,Texture,_,_,_,_,_,_),Texture)]) :-
       data_check :: user_id(S,G,D,O,F) &
source_of_value(ec/1,      [database(farm,soil(S,G,D,O,F,_,_,EC,_,_,_,_),EC)]) :-
                    data_check :: user_id(S,G,D,O,F) &
source_of_value(s_status/1,[derived]) &

prompt(ca_carbonate/1, ca_carbonate, []) &

type(texture/1,nominal) &
type(ec/1,real) &
type(s_status/1,nominal) &
multiple(s_status/1) &
legal(texture/1, texture) &
ll(ec/1,0.1) &
ul(ec/1,10)
}.
```

of database attributes. The developers do not write another code. This shows that the tightly coupling is easy to develop, where the modified inference engine supports dynamically the coupling mechanism. Figure 14 illustrates some knowledge base attributes in soil object that have sources of values in the database.

3.2 Performance Comparison between the Loosely/ Tightly Coupling

The "statistics" command of SICStus prolog is used to measure the run time of the two approaches.

The statistics command displays the standard error stream statistics relating to memory usage, run time, garbage collection of the global stack and stack shifts.

The syntax of this command is:

```
Statistics (? Key , ? Value)
```

This allows a program to gather various execution statistics. For each of the possible keys Key, Value is unified with a list of values. To measure the runtime the following syntax is written:

```
Statistics( runtime, [since start of Prolog, since previous statistics]).
```

This refers to the used CPU time during the execution, excluding the time spent during garbage-collecting, stack shifting, or during system calls. In Muse, these numbers refer to the worker that happens to be executing the call to statistics/2, and so normally are not meaningful.

The comparison between loosely and tightly coupling is made for irrigation subsystem. The irrigation of CITEX is done every month. The user must determine the start and the end of the month of the needed irrigation. In the comparison, the irrigation is run 12 times. The first one is run for only one month, the second is run for two months, and so on. This is done in both versions in the same machine and same database. Only a parameter that is changed is the coupling technique.

We can see that the loosely coupling is better than tight coupling in the length of time at month equal to or greater than three. The database fields that are needed in irrigation subsystem are equal to 13 data fields + 5 fields for each month.

In loosely coupling, the database fields that are collected are 80 fields in all cases. It does not depend on the number of months.

In tightly coupling the database fields that are accessed are given from the following equation:

Number of fields = 13 + (5 * N) where N is the number of month.

The maximum database fields that are accessed = 13 + (5*12) = 73. This number is less than the maximum database fields that are accessed by the loosely coupling. This is because in the loosely coupling all data fields in the view is collected whether these fields are needed or not.

The loosely coupling mechanism accesses the database 17 times only. In each access, it fetches all fields' data in one relation and consequently asserts these data fields in the corresponding knowledge base attributes. In each time of accessing, a relation is retrieved. This reduces the time of database access. The inference engine in tightly coupling accesses the database for each needed database fields. The number of accessing of the database is equal to the number of needed database fields. The minimum access number = 13+5 = 18 times. The maximum access number = 13 + 60 = 73 times.

The time of loosely coupling equals 17 with all cases. The time of tightly coupling is less than the time of loosely coupling in cases of one or two months. It equals 18 and 23 consequently. This difference of time is due to the loosely coupling retrieves all data fields from database whether they are used in expert system session or not. Also, there are some other operations, which are occurred during the collection data process in loosely coupling. These processes are mapping the database fields with knowledge base attributes and asserting them. The time of tightly coupling is greater than the time of loosely coupling in other cases of months. This difference of time is due to the tightly coupling accesses the database for each attribute, but loosely coupling access the database for each relation.

The development of CITEX proved that the TCS is better than the LCS - from expert system development point of view - for the following reasons:

- In TCS, there is no need for write interface program between the database and the expert system, where the interface is already existing.
- No need to map between the database fields and knowledge base attributes.

This means that the expert system developer does not make tedious work when he uses TCS. The developer handles the database attribute as handles the derived attribute in the definition of knowledge base attributes.

Also, the TCS overcomes the dynamic environment changes that make a large problem in LCS. The CITEX and CUPTEX are bi-lingual expert systems (Arabic and English). If the expert system user starts system with Arabic language, and all data are fetched from database using LCS, and if he

changes the application language into English language, a translation problem occurs. In TCS this problem is solved.

Another point of view is the system speed. In this thesis, two equations that calculate the time are derived from the TCS and LCS algorithms. These equations show that the systems speed depends on the ratio of database fields that are used in expert system. The LCS speed is constant in all expert system sessions where all database fields are retrieved in all cases whether they are used during the session or not. In this thesis a real measure of speed of both system were achieved. In this measure, all parameters are the same for the both systems, same machine, same data, and so on. The measure results are equivalent to the two system time equations.

Another point of view is the used memory in both systems. The LCS uses a large amount of memory comparing with the TCS. These due to the TCS only retrieves the required data from database, but LCS retrieves all data from database.

4. Conclusion

In this paper, we have discussed two methods of coupling knowledge-base systems with database systems. These methods are loose and tight coupling. The underlying systems are Loosely Coupled System (LCS) and Tightly Coupled System (TCS). To build these systems we made the following:

- We built an object-oriented layer on the top of the external storage utility of SICStus Prolog language.
- In TCS we modified the searching method of the KROL inference engine.

To illustrate the difference between the two systems, they were used in building two real expert systems that are CITEX and CUPTEX. The development of CITEX and CUPTEX that the TCS is better than the LCS - from expert system development point of view - for the following reasons:

- In TCS, there is no need to write interface program between the database and the knowledge base, where the interface is already existing.
- No need to map between the database fields and knowledge base attributes.

Another point of view is the system speed. In this paper a two equations that calculate the time are drived from the TCS and LCS algorithms. These equations show that the systems speed depends on the total number of database fields that are used in the system..

Another point of view is the used memory in both systems. The LCS uses a large amount of memory comparing with the TCS. These due to the TCS only retrieves the required data from database, but LCS retrieves all data from database.

References

[Al-zobaidie 88] Al-zobaidie, A. & Grimson, J. B. “Use of Metadata to drive the interaction between database and expert system”, Information and software technology, 30, 484-469, 1998.

[Claude 88] Claude Bailly , Paul Y Gloess “combining an expert system with a data base for an application that aids decision-making”, Artificial 1ntelligence, Expert Systems and languages in Modelling and Simulation, © IMACS, 1988.

[Dessouki 93] El-Dessouki, S. Edrees, S. El-Azhary, "CUPTEX: An Integrated Expert System For Crop Management Of Cucumber", (ESADW-93) May, 1993, MOALR, Cairo - Egypt. A. I.

[ESICM 92] ESICM, “Specifications of object-oriented language on top of prolog: KROL.”, Technical Report No. TR-88-024-25 Expert Systems Improved Crop Management. UNDP/FAO. EGY/88/024.

[Heng 97] Heng-Li Yang, “Simple Coupler to link expert systems with database systems, Expert Systems with Applications", An International Journal, Vol. 12, pp. 179-188, Elsevier Science Ltd, 1997.

[Marcel 92] Marcel Holsheimer, “LIFE – WISDOM, A database interface for the LIFE system”, September 1992, Master thesis, Department of Computer Science, University of Twente

[Maria 96] Maria Taboada, Roque Martin, J. Mira, Ramon P. Otero, “Integrating Medical Expert System, Patient Data-Base and User Interfaces", Journal of Intelligent Information Systems, Kluwer Academic Publishers, Boston. 1996

[Nat 88] Nat BaIlou, Hong-Tai Chou, Jorge F. Garza, Won Kim, Charles Petrie, David Russinoff, Donald Steiner, Darrell Woelk “Coupling an Expert System Shell with an Object-Oriented Database System”, Journal of Object Oriented Programming (JOOP) June/July 1988,

[Salah 93] Salah, H. Hassan, K. Tawfik, I. Ibrahim, H. Farahat, "CITEX: An Expert System for Citrus Crop Management", (ESADW-93) May, 1993, MOALR, Cairo - Egypt.

[Selmin 90] Selmin NURCAN, LI Lei, Jacques KOULOUMDJIAN , “Integrating Database Technology and Logic Programming Paradigm”, ACM, 1990.

[Shaalan 98] Shaalan , K., Rafea, M, &.Rafea, A., "KROL: A Knowledge Representation Object Language on Top of Prolog, Expert Systems with Applications", An International Journal, Vol. 15, pp. 33-46, Elsevier Science Ltd, 1998.

[SICS 95] “Sicstus Prolog User’s Manual”, Copyright @1995 by Swedish Institute of Computer Science “SICS”.