IDEA GROUP PUBLISHING

701 E. Chocolate Avenue, Suite 200, Hershey PA 17033-1240, USA
Tel: 717/533-8845; Fax 717/533-8661; URL-http://www.idea-group.com

ITP4910

An Architectural Framework for a Web Based Knowledge System

Sai Kakkaraju
University of Western Sydney, Western Sydney, Locked Bag 1797, Penrith South DC NSW 1797, Australia,
slakkara@cit.ews.edu.au

Vijay Khandelwal
University of Western Sydney, Western Sydney, Locked Bag 1797, Penrith South DC NSW 1797, Australia
v.khandelwal@cit.ews.edu.au

ABSTRACT

In this paper an architectural framework for a web based knowledge system is presented. It is based on an existing web-based system-CBEADS (Component Based E Application Deployment Shell), used for developing and deploying eApplications using traditional database systems. The proposed knowledge based architecture uses logic programming and Datalog database techniques to overcome the drawbacks of traditional database systems. K-CBEADS (Knowledge based CBEADS) distinguishes between data, knowledge, and information. It can generate/capture knowledge and provide higher throughput in information supply which has many uses in knowledge management. Proposed architecture can be used for any web-based knowledge system.

INTRODUCTION

Armour in his article "The case for a new business model: Is software a product or a medium?" [1] Argued that software can be viewed as a medium of choice, which along with brain and books can be used to store knowledge. However unlike the knowledge stored in books knowledge stored in software systems can be executed with resulting advantages.

If we take this view of the software then we need to have a software framework, which can be set up in an organization that is capable of capturing and sorting information (knowledge and data) and providing appropriate information, and access to relevant people dynamically.

Most traditional software development approaches such as the Waterfall method [19], Spiral Model [2,3], Rapid Prototyping etc. rely on being able to completely specify the system during initial requirements analysis. These approaches are well suited for static applications where requirements don't change frequently. Generally the outcome following these approaches is a final product. In dynamic environments like eTransformation, specifying entire requirements initially is not possible as these changes quite frequently with time.

Our focus in this study is on developing a web-based framework, which can be set up in an organization that is capable of capturing and sorting information (knowledge and data) and providing appropriate information/access to the relevant people dynamically.

Such a framework should have following characteristics:

- Ability to distinguish between knowledge, data and information.
- Ability to add new functions and change existing functions while the system is operational.
- Ability to perform inductive tasks to generate and capture knowledge.
- Ability to perform quick searches.

There is much research being carried out currently by logic programming community as well as software engineering community for such a framework. In this paper we propose a new architectural framework suitable for the above purpose by applying logic programming techniques to a web-based system called CBEADS (Component Based E Application Deployment Shell). We call this new architecture K-CBEADS (Knowledge based CBEADS).

This paper is organized as follows. In section 2 we present the existing CBEADS architecture. In section 3 we present the motivation for a new architecture. In section 4 we define data, information and knowledge and provide some technical details needed throughout this paper. In section 5 we present K-CBEADS architecture. Finally in section 6 we conclude this paper.

CBEADS: COMPONENT BASED E APPLICATION DEPLOYMENT SHELL

Component Based E Application Deployment Shell (CBEADS) [9] is a web- based system that runs in conjunction with a web server, and provides a web-based interface to interact with it. As this is a web based application it can easily be deployed as an intranet application within an organization, or as an extranet application linking different organizations.

The shell itself is made up of a number of components. These components can be grouped into two major subsystems. The first is CORE CBEADS that provides the overall framework to which different eApplications can be plugged. It consists of security module, system components, system database and workflow components. The second subsystem consists of various applications plugged into the shell. These consist of application components and application databases. The overall CBEADS architecture is shown in figure 1.

As can be seen in figure 1 all the interactions originating from the users first pass through a security module that carries out authentication. It also checks what applications, and functions within these applications the users are permitted to access.

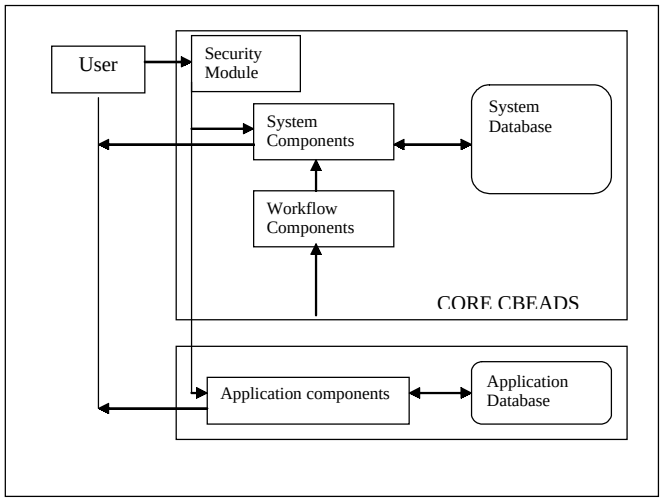
After authentication every user is given a personalized home page. This page displays the applications, and the functions within these applications that the user is permitted to access. As an added security measure before serving each request the security module validates the user authority to access functions involved in serving the request.

Within the system components there are functions to create user groups, and to allocate different applications and functions within these applications to user groups. This, of course, is very similar to the environment in many other application deployment systems.

There is a special function among CBEADS system components that can be used to create new functions. This ability to create new functions enables CBEADS to grow. These new functions can be grouped to form applications.

One can develop and deploy new applications independent of other applications within CBEADS. To develop a new application it is necessary to decide what essential functions are required within that application. For example if the application being developed is a Human Resource Module we need a set of functions to add, edit and delete the

Fig 1: CBEADS architecture



names of employees and other related information. We also need a set of functions to display different reports based on this information such as contact details, salary details, leave taken, etc. All these functions can be grouped to form Human Resources Management application.

The architecture for application databases in CBEADS is very similar to the architecture, which Shaw [20] has classified as "Repository Architecture Pattern". According to Shaw this architecture is suitable for applications in which the central issue is establishing, augmenting and maintaining a complex body of information. In such cases typically the requirement would be for the information to be manipulated in a wide variety of ways.

The other major element in CBEADS is the workflow component. One of the purposes of this component in the case of workflow-based eApplications is to inform users at login time that they have pending tasks to process.

The following summarises the main features of CBEADS:

- It is a Web based system
- New functions can be added and existing functions modified while the system is operational. There is no need to recompile and install the software every time a change is made to an existing function or when a new function is added.
- One can add and change fields in databases, or data structures in data repositories while the system is operational.
- There is a user management and authentication system to allow/ restrict the user accessibility.

At present CBEADS is being piloted in various organizations including at the University of Western Sydney. The applications include stock control system, B2B order processing system, and a teaching allocation system.

MOTIVATION FOR A KNOWLEDGE BASED SYSTEM

It is evident from the above that CBEADS possesses some very desirable features for developing and deploying web based systems. It is built with components, which can be developed and plugged into the system without disturbing the existing system. There is no need for recompilation or shutting down for updating the system. But as it uses a database engine for inferencing it has the same drawbacks as most other database systems. That is,

- It cannot distinguish between data and knowledge.
- It cannot capture knowledge from the users.
- The system can have redundant information in various tables.

Using CBEADS a user can query the system, get a response, create tables, or define relations. But one cannot perform inductive tasks directly. Also it is not possible for a user to get an explanation for an observation. Similarly generating new knowledge from existing knowledge is not possible. Finally there is no way for the system to find new relationships within the defined relations or from within the information provided.

We are therefore proposing a framework where software is considered as a medium for storing knowledge, and in which there are five different ways in for a user to interact with the system [7]. These are through

Observation
Inductive Task
Evidence
Information
Query

To achieve this we are proposing the use of existing logic programming techniques where the terms data, knowledge and information are already defined in terms of logic and Datalog databases (as discussed in the next section), and which can distinguish between data, knowledge and information. The outcome is the proposed K-CBEADS system architecture that preserves the principal characteristics of CBEADS, is capable of distinguishing between data, knowledge, and information, and is capable of creating new knowledge.

DEFINITIONS AND CONCEPTS

In this section we present some definitions and concepts used in this paper. These definitions are mostly standard from logic programming perspective. Some definitions however are slightly modified to suit the context. For a detailed discussion on Datalog databases one can refer to [5]. For definitions of the terms that are not explicitly defined here one can refer to [4,5,16,18].

Data

Data is purely extensional. By extensional we mean that the representation does not denote more than one fact. A *data item* is the representation of one particular fact. In the scope of this paper data items are expressed as Datalog facts. For example, in a *tutorial allocation system* we represent that a tutor (*Sai*), wishes to tutor a particular subject (*KBS*), in a particular campus (*Kingswood*) and at a particular university (*UWS*). We assume that the data items are expressed as tuple of values representing something about the state of the modelled domain in relation with an entity or an occurrence. Thus in Datalog system [4,5,15] the following is a data item.

Tutor (Sai, KBS, Kingswood, UWS). ……*(1)*

Data items are well-formed formulas (wff) in Datalog system with well-defined proof and model theories. The concept of Datalog is explained later in this section. To represent a domain of interest, where every fact in the domain is represented as a data tuple (wff), all that needs to be done is to list the data items in the form of wffs that denotes all the facts that denote the domain. Once this listing has been done insertion, deletion, modification, filtering and combining of such a list is straightforward.

Knowledge

Knowledge is primarily intensional. Intensional means that the representation denotes more than one fact, and hence, it says something about many similar facts in the domain of interest. In the case of knowledge a simple enumeration of wffs is not enough to bring forth all the information. As information is implicit one needs to apply an inference procedure to compute its logical completion, which, by definition, contains all the facts inferable from those wffs. In Datalog context, this completion is a finite set of data representations called ground sentences that could ultimately be enumerated.

A *knowledge item* is the representation of many similar facts, in the sense that from a single knowledge item many similar facts can be made explicit by the application of inference rules.

For instance, it is well known that whoever wishes to tutor knowledge based systems (KBS) also wishes to tutor artificial intelligence (AI) unless explicitly stated. The following Datalog rule represents this knowledge.

Tutor (p, AI, c, u) ¬ Tutor (p, KBS, c, u).....(2)

Intuitively this rule means that if a person (*p*) studying at university (*u*) wishes to tutor *KBS* in a campus *(c)* he or she also wishes to tutor *AI* at the same campus.

From *(1)* and *(2)* one can infer the fact *Tutor (Sai, AI, Kingswood, UWS).*

Similar facts can be inferred for every other tutor of whom it has been explicitly asserted that they wish to tutor KBS.

Information

In general an *information item* [7] is, an answer to the query or an explanation to the observation passed to the user, new axioms or updates to the knowledge base, or new observations (facts) to be added, removed or updated in the database. An information item is again a wff. It is either a data item or a knowledge item derived from the current data and knowledge stocks by an inference procedure. Given a set *Knowledge È Data* items, the amount of available information is that superset thereof characterizable from *Knowledge È Data* by application of an inference procedure. Information is therefore considered as knowledge and data by inferential processes. For example, the answers to a request such as 'List all represented facts of a specified kind' constitute information in the form of data. Given the request ?- Tutor (p,-,Kingswood,-) , a Datalog evaluation engine would list all the names of tutors who would like to be the tutors in Kingswood campus, among which is Sai. If the rule expressing the relationship between KBS and AI suggested above were available, then given the request?- tutor(p,-,Kingswood,-), a Datalog evaluation engine would also provide the information that Sai is a candidate to tutor for AI even though it was only ever explicitly asserted that Sai wants to be the tutor for KBS.

Datalog

A *Datalog* language is any first order language without function symbols. Any term in a Datalog language is either a variable or a constant. Datalog programs have restrictions on clauses that can be formulated using the notions of a safe variable clause and range restricted clauses.

Let $C=(A\neg L_1,..., L_n)$ be a normal clause. Then if L_i is a positive extensional or intensional literal the each variable occurring in L_i is safe or if L_i occurred in an equation where S is isomorphic to T and either if all variables of S is safe or all variables in T are safe then each variable occurring in L_i is safe.

C is range restricted if each variable in C is safe. A database is range restricted if every clause in it is range restricted. A goal is range restricted, if so is the clause A ¬L, where A is any ground atom.

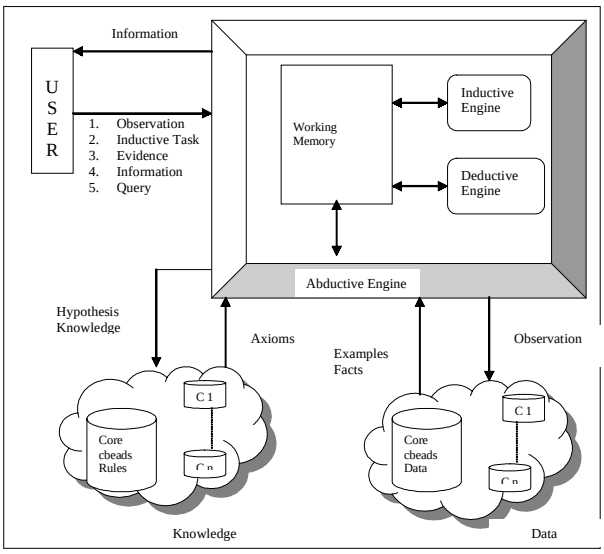
A *Datalog database* is any range restricted database in a Datalog language. A Datalog program P is any Datalog database such that for every clause C ÎP, the predicate symbol in the head of C is intensional.

K-CBEADS ARCHITECTURE

Based on the above, a knowledge based architecture for CBEADS has been developed shown in figure 2. In this architecture we use a knowledge base, a database and three different inference engines. Knowledge base is made of knowledge components from core CBEADS as well as the knowledge part of the working components. Similarly database is made of data components from core CBEADS as well as the data part of the working components. In the broad context of this discussion, following two procedures for computing information from data and knowledge representations are usually considered. In the absence of knowledge representation, abductive engine or inductive engine, this system uses traditional query answering algorithms with traditional databases and works similar to the existing CBEADS system.

K-CBEADS uses epistemic logic programs to represent data and knowledge. In the presence of knowledge representations, K-CBEADS use Datalog evaluation algorithms as in deductive database engines. K-CBEADS system can distinguish between data, knowledge and information. When a user inputs a task (Observation, Inductive Task, Evidence, Information or Query, as discussed in section 3) the abductive engine receives it and places it in the working memory along with relevant axioms from knowledge base or examples from database. Based on the input type either deductive engine or inductive engine or both work with the information in working memory to perform the given task.

Figure 2: Knowledge Based CBEADS Architecture



The major tasks taken care of by the abductive engine are:

- Receiving the input from the user and abducting the necessary information from the knowledge base, or database, and placing them in the working memory.
- Resolving any conflicts.
- Updating knowledge base and database, and passing the information to the user.

Abductive engine proposed in [11,12] amply serves the purpose.

Once the abductive engine places the input and the information relating to the input in the working memory, deductive engine works with the working memory to generate new information.

The major task of the deductive engine is to generate new information (such as answering a query) from the information available from the working memory. Many such Datalog evaluators are proposed [4,5,15,16].

To generate knowledge we use inductive engine with the information in the working memory. Any one of the Inductive Logic Program engines proposed in the literature [13,17,18] extended to suit epistemic logic programs serve the purpose.

CONCLUSION

In this paper we have presented an architecture for a web based knowledge system. It is based upon the idea that "Software is a medium for storing knowledge". It uses three types of inference mechanisms-induction, deduction and abduction. It inherits the merits of the existing web based system architecture and overcomes its shortcomings by having a Datalog database instead of a normal database. This new architecture is capable of creating new knowledge, providing a faster search engine and a good decision support system as it keeps the user well informed by explanations for every observation. Though most of the systems explained here (Abductive logic programming, Inductive logic programming, Deductive databases, Datalog databases, and Component based systems) have been dealt with independently, to our knowledge this

is the first work to combine all these techniques to propose a unified framework.

There are a number of challenges involved in constructing such a framework. The main difficulties are choosing suitable knowledge representation, and updating the existing inference engines to work with the chosen knowledge representation.

We considered epistemic logic programs [8] as the knowledge representation tool and updating the abductive engine [11,12] to suit epistemic logic programming. Future work includes choosing and updating deductive and inductive engines to suit epistemic logic programming.

REFERENCES

[1] Armour, P.G. (2000), "The case for a New Business Model: Is software a product or a Medium". ACM Communications. Vol 43.

[2] Boehm, B. (1988), "A Spiral Model for Software development and enhancement". IEEE Computer May 1988.

[3] Boehm, B. and A.Egyed (1998), "Using win win Spiral model: Case study" in IEEE computer vol July 1998, pp. 33-44.

[4] Ceri, S., G.Gottlob and L.Tanca (1990), "Logic Programming and databases". Springer-Verlag. ISBN 3-540-51728-6.

[5] Das.S.K (1992), "Deductive Databases and Logic programming". Addison-Wesley. ISBN 0-201-56897-7.

[6] Epner.M (2000), "Poor Project Management Number-One Problem of Outsourced E-Projects". Research briefs, cutter Consortium, Vol 7, November 2000.

[7] Fernandes, A.A.A (2000), "Combining Inductive and Deductive Inference for Knowledge Management Tasks". In 11th Intl.Workshop on Database and Expert systems Applications. IEEE Computer Society.

[8] Gelfond.M (1994) , "Logic programming and reasoning with incomplete information", In annals of mathematics and Artificial Intelligence 12, PP:89-116.

[9] Ginige.A (2002), "New paradigm for Developing Evolutanory software to support E-Business" in Hand book of software engineering and Knowledge Engineering, Vol.2, S.K.Chang Ed, World Scientific, PP 711-725.

[10] Ginige.A (2002), "Web Engineering: Managing the complexity of web systems development", Presented at SEKE02, Ischia, Italy.

[11] Lakkaraju, S.K. (2001), "A SLDNF based Formalization for Updates and Abduction", M.Sc (honours) Thesis, University of Western Sydney.

[12] Lakkaraju, S.K. and Yan Zhang (2000), "Rule Based Abduction", In Proceedings of 12th ISMIS, pp525-533.

[13] Lavra

N. and S.D•eroski (1994). "Inductive Logic Programming: Techniques and applications", Ellis Horwood.

[14] Loke S.W and Davison, A. "Logic web: Enhancing the Web with Logic programming", Journal of logic programming.

[15] Loyer Y.N. and D.Stamate (1999). "Computing and comparing semantics of programs in four valued logics". In MFCS99. PP 59-69.

[16] Minker, J. (1996). "Logic and databases: A 20 year retrospective". LID'96. PP 3-57.

[17] Muggleton S. and L.De Readt (1994). "Inductive Logic Programming: Theory and Methods". Journal of Logic programming 19/20 PP: 629-679.

[18] Nienhuys-cheng S.H and R. DeWolf. "Foundations of Inductive Logic Programming", LNAI 1228. Springer-verlag,1997. ISBN 3-540-62927-0.

[19] Royce, W.W. (1970) "Managing the development of large software systems: Concepts and Techniques", WESCON.

[20] Shaw M. and D.Garlan (1996), "Software architecture - Perspectiove on an emerging Discipline", Prentice Hall.