Precision in Design Reusability using Software Agent based Design Triggers

Vinay Goyal
Assistant Professor
Panipat Institute of Engineering & Technology
Samalkha, Panipat (HR.) INDIA

Ashok Kumar
Professor
Department of Computer Sc. & Applications
Kurukshetra University, Kurukshetra

ABSTRACT
Conventional software development methodologies for today's software developments have not been proven good enough to cope with the unconventional demands and rapid application development. There is a serious need for computer aided methodology which can cope with the challenging need of the today's requirements. Computer aided software agents playing important role in new age of software development. The software design phase development experience of software development lifecycle can be enhanced using design triggers agents. This can result in increased efficiency in overall software development process and can help in producing quick solutions resulting in reduced development cost and effort.

General Terms
Agent based systems, Agent oriented software engineering, Design agents, Design triggers.

Keywords
Agent based systems, Agent oriented software engineering, Design agents, Design triggers.

1. INTRODUCTION
The categories and classification of the software requirements is continuously changing over the time in computer history and so are the software development methodologies. The conventional requirement gathering techniques get more advanced when the nature of project become more wide and complex. Design methodologies also require a touch of excellence to cater the cases where old design becomes no more of great usage [2]. The consideration of design pattern is the most advance topic to be considered in the design phase of software development life cycle [7]. It has been identified that the nature of problem statement in the SRS can be categorized into some of known field of domain.

The design statements have been identified on the basis of the domain of SRS problem statements. The design solution frames have been identified; its domain pattern is followed for bridging the problem domain with solution domain. This lead to a planned and rapid development after following the design patterns which get matured over a period of time and over the time they get more organized with heuristics. Finally, it was observed that every software development assignment is having a well defined problem domain whose solution can be identified in parallel in design patterns. One can say that design patterns are the recurring solution to design problems.

As there is a need to bridge the domain gap, the software agents who are also actively associated in the software development lifecycle can play a vital role. Research shows that it is technically possible to associate the software agents for strong assistance for requirement analysis phase [8][12]. The key point is the domain of the software development. For recurring problem statement in a well defined domain, software agents have been used to identify the importance of requirement entities. Agents ensure that all the important aspects have been analyzed and there is nothing left, which is important. This is the most important assurance which any system analyst is always in need of.

On the parallel track, for software design phase, the software agents can be helpful in identification of the most valuable design for a particular (problem) requirement in some well defined domain of solution. The design pattern serves as recurring solution to design pattern, but this could be added with design agents to make a shot in arm. The design pattern or design repository [10] for a well defined problem statement category can be made more useful with the association of design agents which will assist in selecting the best design out of available design on the basis of some well defined parameters.

2. RELATED WORK
Agents and current information technology tools are capable enough to explore trends and patterns of information in plenty from many different heterogeneous sources. Once the information has been gathered in a repository, the user can go through this information and can extract whatever information they are seeking. An agent can operate in a data warehouse discovering information of domain interest. A 'data warehouse' brings together information from lots of different heterogeneous sources. "Data mining" is the process of looking through the data warehouse to find information of interest that you can use to take further actions such as ways to increase sales, find appropriate marketing strategy, etc.

'Classification' is one of the most common types of data mining, which finds patterns in information and categorizes them into different classes. Data mining agents has the capability to perceive major shifts in trends or a key indicator and can detect the presence of new information and can make alert notifications automatically, based on the domain requirements of the user. For example, the agent may detect a decline in the construction industry for an economy; based on this relayed information construction companies will be able to make intelligent decisions regarding the hiring/firing of employees or the purchase/lease of equipment in order to best suit their firm. [6][11].

One of the widely acclaimed architecture is the Brook's subsumption architecture [10]. This architecture is used to model physical robots to bridge perception to action. The same architecture can be used as basis to pure reactive software agents. The architecture consists of a set of modules, each of which is described in a subsumption language based on augmented finite state machines (AFSM). An AFSM is triggered into action if its input signal surpasses some threshold value, though this is also dependent on the values of suppression and inhibition signals into the AFSM. Unlike the classical Artificial Intelligence work, AFSMs represent the

only processing units in the architecture without usage of any kind of symbols [9]

The layered approach is used to place the grouped modules which work in asynchronous manner such that modules in a higher level can inhibit those in lower layers. Each layer has a hard-wired purpose or behavior or in agent domain, we can term it as agent goal, e.g. to avoid obstacles or to enable/control wandering as shown in figure 1. This architecture has been used to construct ten mobile robots at MIT. Steels use similar agents to Brooks in order to investigate cooperation between distributed simulated robots using self-organization [10].

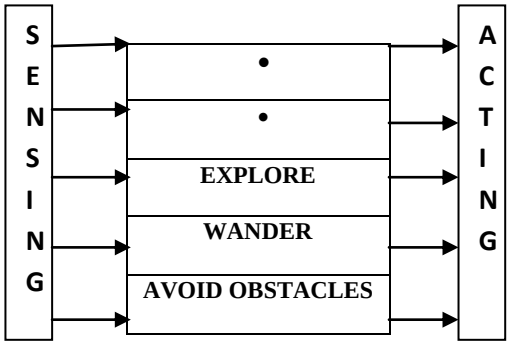

Fig 1: ***Brooks Subsumption Architecture***

The most basic reactive architecture is based on situated-action rules. The work done by Suchman provides basis to this kind of architecture. Positioned action agents act fundamentally in ways which is appropriate to its position, where position refers to a prospectively compound combination of internal and external events and states. Positioned-action agents have been used in PENGI [22], a video game.

Scientist and Researchers at Philips research laboratories in UK have implemented a position-action based language called the RTA programming language. This language has been used to implement characters in computer games. Wavish & Graham [23] have proposed another language based on a modal logical formalism, which in turn is based on a paradigm called situated automata. Agents written in this language are compiled into digital circuits which implement the reactive agent system.

Though the agents discussed in this paper are termed as collaborative, deliberative agents; they may not have been fully collaborative as defined, but they were in character. For example each agent in distributed vehicle monitoring system is a blackboard knowledge source whose task is to identify the vehicles track from acoustic data.

As each of these agents shared a global pool of knowledge for problem solving, these agents can be considered as purely autonomous and moreover cooperation between them is also considered as basic as all the agents share knowledge from the common knowledge pool. The other works such as MACE [15], MCS [16] and IPEM [17] have deliberative agents with non- linear planning modules that support the mutual cooperation among themselves in their operating environment. Other planning-based prototypes include Hayes-Rothi's GUARDIAN architecture [18].

At BT Labs, two prototype collaborative agent-based systems have been developed recently: the ADEPT and MII prototypes. ADEPT [19] employs collaborative agents in the application area of business process re-engineering (BPR) while MII [20] demonstrates that collaborative agents can be used to perform decentralized management and control of consumer electronics, typically PDAs or PCs integrated with services provided by the network operator.

High Level Logic (HLL) is a framework for intelligent applications that is easy to understand and use, even by less experienced programmers. Applications consist of integrated chains of decisions and operations. Development of cooperating systems, each of which is built on HLL, is as easy as building a single integrated system.

Application programmers build protocols that link to specialized application components. HLL "message passing" is taken to the extreme, as with independent interacting agents, while providing the backbone for complete applications around which specialized application components are applied and interact. Individual actors (intelligent agents) within the framework have specialized roles and responsibilities (and authorities) that are designed to complement one another as shown in figure 2.

The HLL components form a complete working organization that controls, manages, and executes the "chains of decisions and operations" of the application.

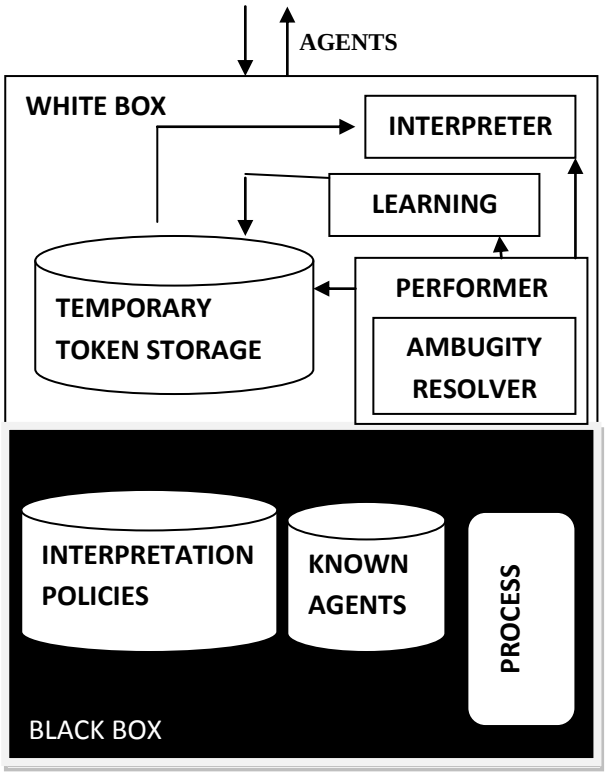

Fig 2: Agents and interpretation policies.

Today, the concept of software agents that perform tasks previously performed by specialized humans is getting popular – such as an Internet salesperson. It also includes automated telephone answering systems that try to figure out what you want and direct your call, perhaps to another automated system.

In the era of 90s, it was not difficult to realize that using current technology; it is possible to create a much more powerful system much more easily than originally conceived

more than a decade earlier. Java's network support for example (and the Internet of course), provides powerful ways for components to interact even when they are physically on systems half a world apart (or in outer space). Large enterprise systems (well supported by Java EE) can provide systematic interaction between departments and operations [13].

Agents add value to the traditional software design by offering tools for handling the most general level of the problem domain. They represent the main building blocks of a distributed software system without describing the internal structure of the individual blocks as shown in figure 3 [3].

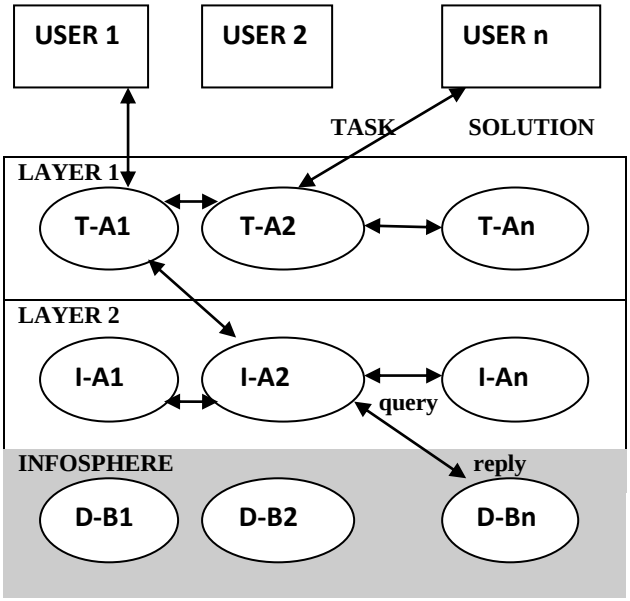

Fig 3: The Pleiades Distributed System Architecture

To build generic solutions to one-off problems is a tedious task for the software developers especially in the agent community. A tailor made design is mostly required for specific problem domain rather than devising an architecture or test bed that apparently enables a whole range of potential types of system to be built. In such situations, a custom built solution will be easier to develop and far more likely to satisfy the requirements of the application [4].

The Unified Modeling Language (UML) is gaining wide acceptance for the representation of engineering artifacts in object-oriented software. Our view of agents as the next step beyond objects leads us to explore extensions to UML and idioms within UML to accommodate the distinctive requirements of agents. The result is Agent UML (AUML). [1]

This subset was chosen because interaction protocols are complex enough to illustrate the nontrivial use of AUML and are used commonly enough to make this subset of AUML useful to other researchers. Agent interaction protocols are a good example of software patterns which are ideas found useful in one practical context

A specification of an AIP provides correlation that might be used to solve problems in system analysis and design. Increased attention is paid by industrial and business users over the results provided by agent's community. These results can be quickly and correctly implemented in the practical applications if the research work is done in accordance with the modern software engineering practices. AUML builds on the acknowledged success of UML in supporting industrial-strength software engineering. The idioms and extensions proposed here for AIP's—as well as others that the researchers are developing are also contributing to this objective [5].

3. PROBLEM STATEMENT

Most of the design models followed in modern age of software development are not having the active intelligence or assistance which can guarantee the successful and accomplishment of the implementation of the requirements. Even, the availability of the design patterns for a domain specific only provides the list of possible solution for a problem statement. There is a need of the domain specific knowledge agent for the applicability of design patterns efficiently.

The design reusability can also be achieved if there is some mechanism available which can assist in selecting the best design module out of available modules. In either case, be it a design pattern, or reusable design, some monitoring application is required, that can suggest the best design item for defined problem. The monitoring application will proactively give the best decision on the basis of previous experience for similar use-case and aligned domain.

This indicate towards the need of an agent which can trigger for the best design in available design repository and can assist the user in selecting the most suitable design pattern out of available design pattern. The lack of agent assistant results in the slow process of software development life cycle. If the design phase can be assisted with the agent based approach, it will result in reusable design, which in turn, results in spending very less or no time in software development. This is because the design module which is selected from the available modules is already having the code/program modules.

The design triggers are required which will proactively work in association to identify the best match for the problem domain inputs. While the user will be looking into the repository and search for some of required design modules, design triggers [trigger, because it will initiate itself on user activated event of search process] initialize the agents. The attribute matching for the design lookup from repository and pattern search initialization within the design pattern statement need to be self initialized without the user dependency.

We need a reliable approach to tackle the problem of overall time saving for any SDLC activity. The reusable design, with associated code available, and the available code is tested as well, there is no need to write fresh test cases for newly developed module. This kind of reusable environment is required which can assist for all stages of SDLC.

The UML design details are project oriented. They need to be defined on the problem statement. There have been some initiative taken by researchers to make UML more useful to designers and programmers but even after UML definitions for problem based namespace, they have not added much value to overall system. This is because of the reason that there is no system available which can search in UML modules and give the value for appropriateness for a given UML design requirement.

UML based design can be made more usable when we add the attribute assignment to UML design item. On the top of

design items, there will be agent which will take care of the requirement and proactively and collaboratively start its gears to produce valuable information which will be useful for reusing UML by the system designer.

4. EXPERIMENT METHODOLOGY

The experiment was carried out with design triggers for selecting the best design artifact from design pattern and available reusable design. Nine problem domains have been identified and their attributes were defined for the reusable design modeling using design trigger agents. Similarly, for the design pattern based design approach the attribute were assigned for the known design pattern method.

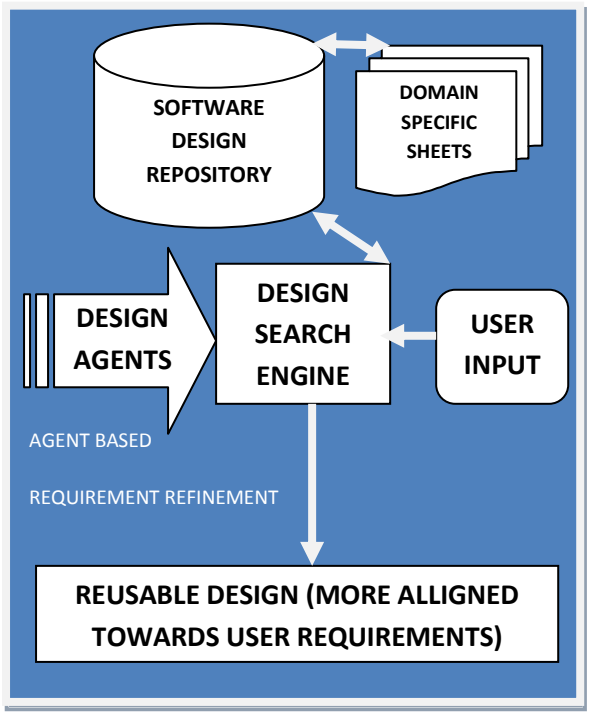

Fig 4: Framework to use the Design Agents as pro-active assistant for design search refinement.

As shown in figure 4, the user needs one design for a problem space. User interacts with the system to look for a design. The observer agent (design agent) will monitor for the inputs supplied by the user, without prompting him, and automatically starts the design search triggers. This starts the searching mechanism for design lookup on the basis of reusable design and the design pattern similarity search.

This process is an added advantage to traditional search where user will never prompted for best design recommendations automatically. The recommendation and guidance for design selection also show the scale of appropriateness of the design item in the specified scenario. The results are very much useful for the user and help him in concentrate on the most suitable designs available.

The domain specific design sheets are stored in software design repository. This is a collection of the design elements arranged on the basis of category in order to follow up with the design patterns. Every design element is having a set of keywords which will be used for the search process. The domain specific layout plays an important role with design triggers. When user input the requirements, the design triggers take the values from user and start a parallel thread for providing a better solution on the top of plane/un-intelligent search. The design triggers filters the results by running a second process of meta-search which arrange the results on the basis of appropriateness and clear down all results which are not associated which the user query.

The experiments are carried out in two separate sessions. The one setup was using the agent based triggers to assist the search process. The second session was just simple look up process without any assistance. The same inputs were passed to the two setups. The outcomes of the design search were compared with using design triggers and without using design triggers. The difference shown was substantial which uses the design triggers. The appropriateness while using the design triggers was much better.

The time taken in identifying the best results for design is also less which improves the user's overall experience. The user feedback for further refinement is collected in interactive system which is also handled by the feedback agent. This is two way interactions, where user is specifying his details and agent feedback agent is providing the related and most appropriate modules after interacting with the basic design trigger search engine.

Table 1: Improvements in user experience in design search results using design trigger agent.

Simulation run	Reusable design type	User experience with Agent based System (Time in min)	User experience without Agent based System (Time in min)
1	Event Scheduler	19	150
2	Part search	16	78
3	Report logger	23	67
4	Text reader	16	34
5	Billing	20	56
6	Bar code generator	13	27
7	PC Monitoring system	15	78
8	Contact book	46	102
9	Auto inventory reporter	54	133

Table 2: Improvements in the appropriateness in the design search results using design trigger agent.

Simulation run	Reusable Design Detail	Level of appropriateness with Design trigger system	Level of appropriateness without Design trigger system
1	Event Scheduler	35	10
2	Part search	56	16
3	Report logger	51	13
4	Text reader	33	8
5	Billing	67	13
6	Bar code generator	87	17
7	PC Monitoring system	33	19
8	Contact book	22	7
9	Auto inventory reporter	25	9

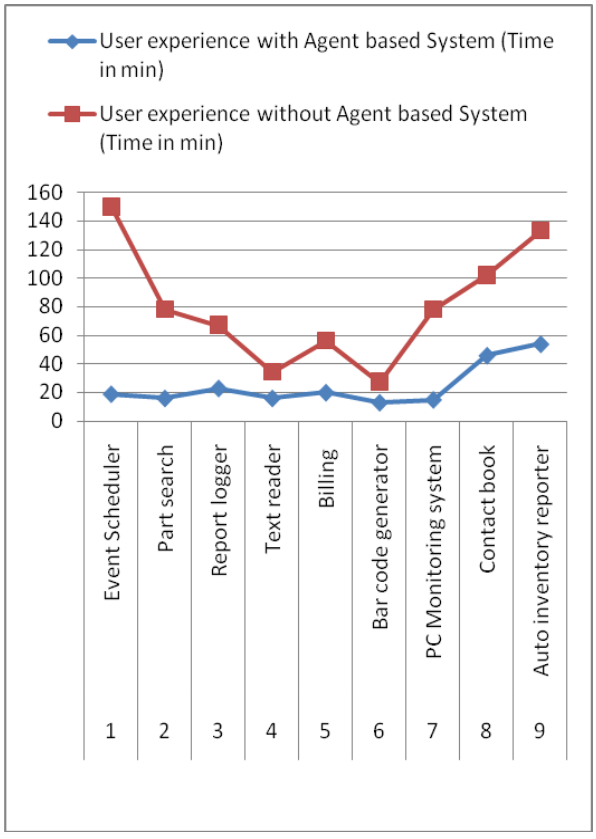

Fig 5: Improvements in user experience in design search results using design trigger agent.

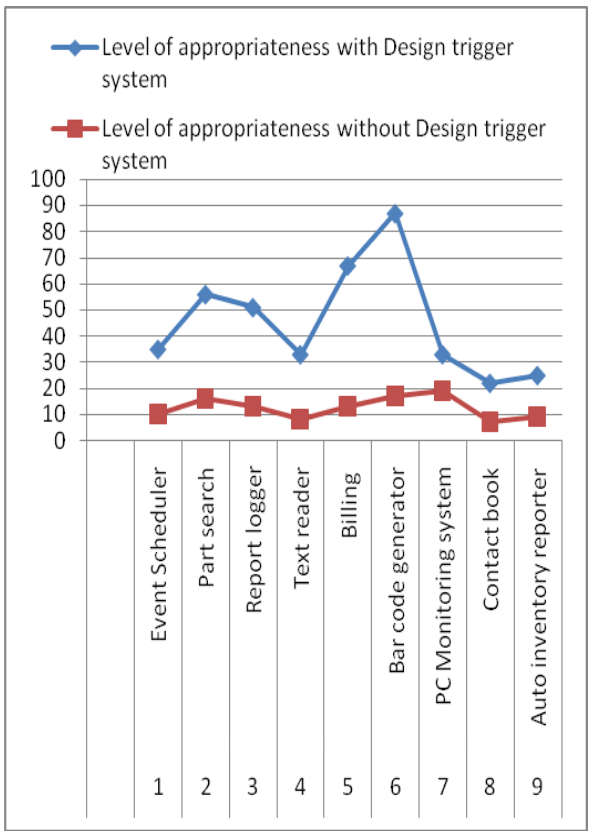

Fig 6: Improvements in the appropriateness in the design search results using design trigger agent.

5. RESULT & CONCLUSION

The reusable system can be useful if they can be easily identified, can easily be fitted in to the required domain. The term reusability can be considered as a benefit for the development of a system if it saves time, resources and speed up the final deliverable product. However, the reusable system are hard to implement and to identify the reusable module which can be best fit in a required domain requires a lot of time and efforts.

The experiment carried out to identify the reusable system using agent based triggers. The experiment shows that the system solved the problem of identification of best fit solution up to some extent. The agent based triggers work on the result set of first level query output. User provides the input to search the reusable design module, and the design trigger will further match the result with the problem domain. This act will minimize the result set and users have to spend less time to find the reusable module.

Table 1 shows the level of reduction in time spent (in minutes) to find a reusable module. The reduction in the amount of time to get the final reusable module is considerably much lesser as compared to the agent-less system. The design trigger agent refines the search for Event scheduler by matching its category to its domain of 'Planner'. When the design element is identified by design trigger, it matches its fitment in the requested domain by checking the subsequent keywords supplied.

The figure 5 shows the overall improvement in the time reduction which provides the enhanced overall solution to the problem and hence improves the user experience. As per our experiment, we have considered the user experience as how much time is being spent to get the valuable results. Less time taken for the search result means user experience is very good.

The time spent on the results obtained is not the only the sole parameter. The design search process should produce the most appropriate results with great precision. When we have used the design trigger based search, we have found that the search results are less in number and they are closely related to the problem domain for which the reusable solution is required as shown in table 2. Figure 6 shows the level of improvement in the appropriateness in the design search results.

All elements that make up the control-flow of a particular agent are grouped under the common concept, making it easier to identify larger units of the program that belong together semantically. This is a key factor to use the pro-active agents to assist the user for taking the decision in domain specific problems. It supports the principle of locality even better then the object-oriented view does. In object-oriented systems, the control-flow specification is spread all over the entire program code. The agent-oriented view introduces a novel approach for conceptual grouping of software design components with some well defined bounding. These well defined bounds help in assignments of the search targets with satisfactory precision level.

Agents can be very helpful in the decision making assistance process. The proactive agents can work in background and they can be tuned to run in a parallel mode, which user may be in need of as an assistance to accomplish a task. The results of the normal process can be compared with the agent's process and the best possibilities can be generated. The domain specific problems get more refinements. Agent works in a co-operative and autonomous mode and this is the most desirable task for the query search and shortening the domain specific results without active or direct human intervention. Agents are found useful in our experiment and the perfection of reusability is achievable to a great extent.

6. REFERENCES

[1] Bryson, Joanna, and Brendan McGonigle, "*Agent Architecture as Object Oriented Design,*" Intelligent Agents IV: Agent Theories, Architectures, and Languages.

[2] S. Matsuoka and A. Yonezawa. *Analysis of Inheritance Anomaly in Object-Oriented Concurrent Programming Language*s. In G. Agha, P. Wegner and A. Yonezawa, Eds. Research directions in Concurrent Object-Oriented Programming, pp. 107-150. The MIT Press, Cambridge, MA, 1993.

[3] Henry A. Kautz, Bart Selman, Michael Coen, Steven Ketchpel, and Chris Ramming, *An Experiment in the Design of Software Agents*, AI Principles Research Department AT&T Bell Laboratories, Proceedings of AAAI-94, Seattle, WA, July 1994, http://www.cs.cornell.edu/selman/papers/pdf/94.aaai.bots.pdf

[4] Nicholas R. Jennings and Michael Wooldridge, *Agent-Oriented Software Engineering*, London E1 4NS, United Kingdom, http://icc.mpei.ru/documents/00000827.pdf

[5] , James J. Odell, H. Van Dyke Parunak Bernhard Bauer James Odell Associates," *Representing Agent Interaction Protocols in UML*", 3646 W. Huron River Dr., Ann Arbor, MI 48103 USA.

[6] Kundu, S., "*Design of web data mining in agents based e-commerce"*, Computer and Communication Technology (ICCCT), 2011 2nd International Conference on 15-17 Sept. 2011

[7] E. Gamma, R. Helm, R. Johnson and J. Vlissides. *Design Patterns: Elements of Reusable Object Oriented* Software. Addison Wesley, 1995.

[8] Ashok Kumar, Vinay Goyal, "*Software requirement analysis enhancements by prioritizing requirement attributes using rank based Agents*", (IJCSIS) International Journal of Computer Science and Information Security, Vol. 9, No. 8, August 2011, (pp. 105-114).

[9] The Distributed System Architecture for agents, http://agents.umbc.edu/introduction/ao/5.shtml

[10] Brookís Subsumption Architecture, http://agents.umbc.edu/introduction/ao/5.shtml

[11] D. Kerr, D. O'Sullivan, R. Evans, R. Richardson and F. Somers. *Experiences using Intelligent Agent Technologies as a Unifying Approach to Network and Service Management.* In Proc. of IS&N 98, Antwerp, Belgium. 1998.

[12] P.K. Suri, Gurdev Singh, "*Framework to represent the software design elements in markup text - Design Markup Language (DGML)",* International Journal of Computer Science and internet security, Vol. 10 No. 1 pp. 164-170.

[13] G. Lavender and D. Schmidt. *Active Object: An object behavioural pattern for concurrent programming*. In J.M. Vlissides, J.O. Coplien, and N.L. Kerth, Eds. Pattern Languages of Program Design. Addison-Wesley, Reading, MA, 1996.

[14] http://isr.nu/hll/project /JavaNetReflect/

[15] Gasser, L., Braganza, C. & Herman, N. (1987), "*MACE: A Flexible Testbed fo Distributed AI Research*", In Huhns, M. (ed.), Distributed Artificial Intelligence, Research Notes in Artificial Intelligence, London: Pitman, Chapter 5, 119-152.

[16] Doran, J., Carvajal, H., Choo, Y. & Li, Y. (1991), "*The MCS Multi-agent Testbed: Developments and Experiments*", in Deen, S. (ed.), Cooperating Knowledge based Systems, Heidelberg: Springer-Verlag, 240-251.

[17] Ambros-Ingerson, J. & Steel, S. (1988), "*Integrating Planning, Execution and Monitoring*", In Proceedings of the 7th National Conference on Artificial Intelligence (AAAI-88), St Paul, MN, 83-88.

[18] Hayes-Roth, B. (1991), "*An Integrated Architecture for Intelligent Agents",* SIGART Bulletin 2, (4), 79-81.

[19] OíBrien, P. & Wiegand, M. (1996), "*Agents of Change in Business Process Management*", British Telecommunications Technology Journal 14 (4), October.

[20] Titmuss, R., Winter, C. S. & Crabtree, B. (1996), "*Agents, Mobility & Multimedia Information*", Proceedings the First International Conference on the Practical Application of Intelligent Agents and Mullti-Agent Technology (PAAM ë96), London, 22-24 April, 693-708.

[21] Brooks, R. A. (1986), "*A Robust Layered Control System for a Mobile Robot*", IEEE Journal of Robotics and Automation 2 (1), 14-23.

[22] Agre, P. E. & Chapman, D. (1987), "*Pengi: An Implementation of a Theory of Activity*", Proceedings of the 6th National Conference on Artificial Intelligence, San Mateo, CA: Morgan Kaufmann, 268-272.

[23] Graham, M. & Wavish, P. R. (1991), "*Simulating and Implementing Agents and Multiple Agent Systems*", In Proceedings of the European Simulation Multi-Conferencce, Copenhagen, June.