TACKLING SOFTWARE COMPLEXITY USING MODEL DRIVEN MULTI AGENT SYSTEM

P. Felcy Judith
Research Scholar. Mother Theresa University
Kodaikanal, Tamilnadu, India.
Sr. Lecturer, T.John College Bangalore.
felcy_judith@yahoo.com

Dr. E. Kirubakaran
AGM, BHEL
Trichy, Tamilnadu, India
e_kiru@yahoo.com

Abstract—The major challenge of current software development is about increasing the level of abstraction. Model Driven development raises the level of abstraction and models could be used across technologies and platforms. Model driven development also addresses the other challenge of increasing the level of automation. The code generated could be used to build services and integrate services. Even though the models solve some of the challenges in software industry but as the software size increases it forces challenges like Maintainability of models, Skill level, Tools and Technology.

On the other hand agent technology is suitable for handling current software development challenges, more dynamic and instead of acting as a single component acts as a team and provides interactive solution. Lack of modularity in MDA could be solved with agent technology. Whereas the lack of development practices in the agent technology could be solved by MDA

Keywords-Agent; Multi-Agents; Meta model, Modeling Agents; Agent Oriented Software Engineering; ROADMAP; RAP/AOR; MDA; MDD; MDE; DSL; MAS;

I. INTRODUCTION

Software development, as currently practices is slow, expensive and error prone, often yielding products with large numbers of defects, causing serious problems of usability, reliability, performance, security and qualities of service [13]. The software industry needs practices to work on higher level of abstraction to meet the existing and future demand. Not just the level of abstraction, level of automation needs to be increased in software development processes. Software products are in some aspects looks like real world entities like automobile spares, but at the same time it is not physical but logical. The reliability of software depends on its logical qualities like correctness and robustness. The reusability of software component is one of the highest parameter to determine cost reduction, risk reduction, time to market and quality improvements. The current level of automation enables us to hand-stitch services through orchestration to build complex software structures, but to meet the demands of the future we need processes to improve automation like the way the semiconductor industry evolved.

Recent statistical study says “17 percent of large IT projects go so badly that they can threaten the very existence of the company, on average, large IT projects run 45 percent over budget and 7 percent over time, while delivering 56 percent less value than predicted [11]”. These numbers suggests there are still some fundamental issues with respect to software development. After comparing with other engineering disciplines, Jack Greenfield and Keith Short [12] prescribed One-off development or development in isolation, Monolithic system - increasing system complexity, working at low levels of abstraction, process immaturity and rapidly growing demand for software systems as the primary reasons for lack of predictability in software development.

One-off development is outcome of the system which is not designed for reuse. Mostly software components are not reusable within the similar systems. This happens when systems were developed in relative isolation without the vision of reuse of system totally or partially. This generally results in development of similar functionality over and over again within the product and across products. Monolithic systems talks about the long-reaching dependencies and high system complexity resulting in tight coupling in software systems. Monolithic designs often results in cascading changes i.e. changes made in one part of the system trigger changes in many other part of the system raising the complexity of software system. Current software languages and tools work at low levels of abstraction to provide greater flexibility. Even though the third generation languages do justice by increasing the level of abstraction without compromising flexibility, still the level of abstraction is not higher enough to produce reusable code across domain and platforms. Software development processes are not matured enough in comparison with other engineering disciplines to produce flexible and predictable software. Rapidly growing demand of software increases along with the size and the complexity of the software. This brings in the necessity of standardization of software systems like other engineering divisions where two entirely different products could exchange their parts. This research aims at developing DSL based agent models and to generate code for generating MAS software.

II. SOFTWARE COMPLEXITY

The current day software development becoming increasingly challenging because of its complexity. Also the complexity gets multiplied by its distributed nature. End users expect the software to be more fast, reliable and scalable in spite of unpredictable changes [4] within the process of software development. The software industry always tries to develop tools, technologies and process to
narrow the gap between the user requirements to software development. Numerous programming languages have been created and development methodologies were invented for building applications. The object oriented development was aimed to move the software closer to the reality [2] in helping a quick and easy way of creating software. But still with the demand exceeds the ability to produce software and due to this the software gap is gradually increasing in time [5]. Every software transition was motivated in lowering cost and time to market and to raise the quality of the product by applying automation [1]. Over the period of many years software industry is focusing on replacing most of the manual software development methodologies with the automated methods [1]. This could be achieved, by increasing the level of abstraction of software engineering closer to the actual domain in order to bridge the gap [5]. In comparison to other industries like automobile the software development industry has not achieved closer repeatable models for faster development. In order to achieve that need for programming languages to represent the problem domain is needed. These languages need to have more flexibility to the domain in building the domain models and have lesser focus to the technological problem [4].

III. MODEL DRIVEN DEVELOPMENT

Model-Driven Development (MDD) which is an Object Management Group (OMG) trademark [3] has developed methodologies to solve the problems causing Software-gap. The focus of these methodologies is to develop domain-specific languages (DSL) using which user can develop the domain specific models and later produce code and other artifacts from these models [5]. Domain specific languages are designed to solve domain problems and to increase productivity in software development process and to move the requirements closer to the product. Domain specific languages are equivalent to assembly lines in automobile industry [2]. Unlike general purpose modeling languages Domain-Specific Languages (DSL) brings the modeling much closer to the domain experts and enables easier maintenance and evolution of such models which contributes to the desired productivity increase and to the agility of MDD. Even though MDD seems to be an ideal solution still lots of challenges before it-self to develop theoretical and practical development methodologies to make it usable by the software industry [6].

The general view about the models in software development is that the models are used mostly for the documentation and play a secondary role in describing the high-level design ideas. But in MDD models play primary role without missing out the secondary usage expected in the general usage [6]. MDD uses models to represent a system element and their relationships. Models flow as input and output throughout the various development phases [10]. MDD has two main themes, one is to raise the level of abstraction and the other is to raise the level of automation [6]. These themes results in increase of product quality, productivity in development. MDD can also provide automation support on various development activities like debugging, building and deploying with the information collected from models. MDD can essentially provide using domain specific languages to write specifications of software to capture developer intent, to isolate and encapsulate variable points, to automate generation of partial or complete implementation of software artifacts.

Model-Driven Engineering (MDE) which is a broader vision of MDD to enable generative and transformational techniques in software engineering [2]. MDE express domain concepts effectively across platforms. MDE combines DSLs and transformation engines and generators [9]. MDE identifies any software artifacts as Model or an element in a model. Some of the common MDE available in the industry are OMG Model Driven Architecture (MDA) [7] and Microsoft Software factories [8].

IV. MULTI AGENT SYSTEMS AND AGENT METHODOLOGIES

Multi Agent Systems (MAS) is based on distributed artificial intelligence computing, where the aim is to split a complex problem into several subtasks and distribute the management of these tasks to individual software entities [15]. This allows system intelligence to be distributed across the system components rather than being concentrated on a single point. “An agent is a computer system, situated in some environment that is capable of flexible autonomous action in order to meet its design objectives [14]” this is the commonly accepted definition for agents. An agent control the environment through sensors, actuators and possess well defined boundary. Agents are flexible by exhibiting the following characteristics [17].

Reactive – Agents were able to perceive their environment and respond to the changes occur in their environment in order to satisfy their design behavior

Pro-Active – Agents exhibit goal-oriented behavior by taking initiative in order to satisfy their design behavior

Being Social – Agents communicate with other agents in order to satisfy their design behavior

“Multi-agent systems (MAS) are computational systems in which a collection of loosely coupled autonomous agents interact in order to solve a given problem. [16]” Agents communicate, cooperate, coordinate and negotiate for

1. Achieving a common goal
2. Monitor the progress of the team effort
3. Help another in need
4. Co-ordinate individual action so that they do not interfere with one another
5. Communicating success and failures
6. To establish zero competition among team members

After careful analysis of Agent oriented methodologies, combined model of ROADMAP (Role-Oriented Analysis
and Design for Multi-agent Programming) and RAP/AOR (Radical Agent-Oriented Process/Agent-Object-Relationship) has closer viewpoints to MDA [18]. The ROADMAP methodology was originally proposed as an extension of the first version of Gaia [19]. The RAP/AOR was originally proposed by Wagner [20], the combined methodology of ROADMAP and RAP/AOR was introduced in 2006 [21]. The ROADMAP adds emphasis on domain and system analysis. The ROADMAP models are vertically classified into domain-specific models, application-specific models and reusable services models. Domain specific model defines the environment model and knowledge model. The application specific model describes Goal model, Role model, Agent model and Interaction model. The reusable services model describes Social model and Service model. The RAP/AOR methodology is meant to create models like Business process and Supply chain management. This methodology is based on Agent-Object-Relationship Modeling Language (AORML). These models could be used to model agent perceptions, beliefs and commitments. Agent interaction like commitments and claims between two or more agents could be depicted in AOR Interaction-Frame diagram.

V. IMPLEMENTATION HIGHLIGHT

1. Designed software artifacts using combined ROADMAP and RAP/AOR agent oriented methodology on the outsourcingdomain [31].
2. Developed graphical DSL to support agent oriented software design [32].
3. Using the graphical DSL to design graphical models for the outsourcing domain [32].
4. Developed KQML based communication protocols for agent communication [33, 34].
5. Generate code to prototyping extent from the designed models [32].
6. Validating the study to prove the DSL based agent development provides flexibility in software development and equally increases the level of abstraction and automation.

VI. VALIDATION

To validate the design and implementation two methodologies were adopted. One to prepare a questionnaire to record observation from the developers to first record their understanding on software complexity, later to provide short training on the DSL based agent development and to make them re-evaluate the similar metrics for measuring complexity. Secondly to consider few open source code and calculate various software metrics like cyclomatic complexity, depth of inheritance, lack of cohesion, instability etc., and to compare with the metrics of the agent based open source software. In the first methodology mainly the software is classified into tiny less than 100 story points, small less than 1000 story points and Medium/High which is greater than 10000 story points. Also the measurements were made on average iteration, average time taken for a single iteration, average unit testing bugs, average system testing defects, average design issues observed, average integration issues observed and finally overall maintenance complexity. The results were given in figure 1 for measuring maintenance complexity. The results show the development approach prescribed impacts greatly in the Medium and High sized application, where the software complexity is very high.

VII. CONCLUSION

This research aims at the challenges of the software development addresses solutions to raise the level of abstraction and level of automation by using models and also increases the life time of the software. To provide solution to the modularity issues generally with the software development and specifically to the modularity issue with the usage of models, multi agents were used. A detailed development methodology is introduced for using agents and specific graphical DSL were created to design multi-agent models. These distributed multi agent models could be integrated to generate code for developing agent based software products. These prescribed methodologies were
designed and implemented in the outsourcing management domain. Finally the proposed methodologies were validated with a survey and proved the application of these practices helps in reducing the software complexity from very high in medium/large application to medium and high. Also the choice of agent technology to reduce the complexity is proven by analyzing software metrics of few open source software’s. With little of improvements the agent technology could be made practical for industrial usage.

VIII. REFERENCES

[30] Domain Specific Languages, Martin Fowler, Addison-Wesley Professional, September 2010