Importance Of Different Agents in Software Development

P. S. K. Patra

Prof. & HOD, Department of CSE, ACT, Chennai, TN, India

Abstract—The tendency of increasing technical complication of software development when tied with the need for knowledgeable and predictable progressive process using agents and multi-agents has driven software developers to establish innovative system development models. The upcoming industry/organization demands the need to automate its various day-to-day activities using different life cycle models to develop software. Due to availability of some standard structural methodology that can be introduced in various fields so that the transition from manual to automated system became easy. A number of models like waterfall, prototype, rapid application development, V-shaped etc. are emphasized in this paper towards building new or improved system In this paper, our basic focus on the comparative analysis of these Software Development processes and usage of agents and multi-agents in different activities of software development.

Keywords: Agent, Multi –Agent, OMG, UML & MAS etc.

1.INTRODUCTION

Software engineering techniques are a key prerequisite of running successful software projects. A *software methodology* is typically characterized by a *modeling language* (used for the description of models, and for defining the elements of the model a specific syntax or notation (and associated semantics) and a *software process* defining development activities, interrelationships among activities & ways in which different activities are performed.

- Deploying agent technology successfully in industrial applications requires industrial-quality software methods and explicit engineering tools.
- Agent technology has still not met with broad acceptance in industrial settings (despite some encouraging success stories).
- Three characteristics of commercial development have prevented wider adoption of agent technology:

scope of industrial projects is much larger than typical research efforts skills of developers are focused on established technologies, not leading-edge methods and programming languages use of advanced technologies is not part of the success criteria of a project Methods for commercial development must depend on widely standardized representations of artifacts supporting all phases of the software lifecycle. Currently, technologies in use by industry (e.g. the Object Management Group's (OMG) Unified Modeling Language (UML) accompanied by process frameworks such as the Rational Unified Process), cannot cope with the required modeling artifacts for agent technologies.

2.EXISTING SYSTEM

Most early approaches supporting the software engineering of agent-based systems were inspired by the **knowledge engineering** community.
Agent-oriented approaches focus directly on the properties of agent-based systems and try to define a methodology to cope with all aspects of agents. A relatively new tendency is to base methodologies and modeling languages on **object-oriented**

techniques, like UML, and to build the agent-specific extensions on top of these object-oriented approaches.

Knowledge Engineering Approaches - MAS-CommonKADS

- CommonKADS is a knowledge engineering methodology as well as a knowledge management framework
- The CommonKADS methodology was developed to support knowledge engineers in modeling expert knowledge and developing design specifications in textual or diagrammatic form.

MAS-CommonKADS Layeres

- The *Organization Model* describes the organizational context in which the knowledge-based system works (knowledge providers, knowledge users, knowledge decision makers)
- The *Task Model* describes the tasks representing a goal-oriented activity, adding value to the organization, and executed in the organizational environment
- The *Agent Model* describes all relevant properties like various roles, competencies and reasoning capabilities of agents able to achieve tasks of the task model.
- The *Knowledge Model* or *Expertise Model* describes the capabilities of an agent with a bias towards knowledge intensive problem-solving capabilities.
- The *Communication Model* describes — in an implementation independent way — all the communication between agents in terms of transactions, transaction plans, initiatives and capabilities needed in order to take part in a transaction.

The *Design Model* describes the design of the system, its architecture, implementation platform and software modules.

Apart from that the following table illustrate the different features of different existing soft ware development life cycle model.

Features	Original water fall	Iterative water fall	Prototyping model	Spiral model
Requirement Specification	Beginning	Beginning	Frequently Changed	Beginning
Understanding Requirements	Well Understood	Not Well understood	Not Well understood	Well Understood
Cost	Low	Low	High	Expensive
Availability of reuseable component	No	Yes	yes	yes
Complexity of system	Simple	simple	complex	complex
Risk Analysis	Only at beginning	No Risk Analysis	No Risk Analysis	yes
User Involvement in all phases of SDLC	Only at beginning	Intermediate	High	High
Guarantee of Success	Less	High	Good	High
Overlapping Phases	No overlapping	No Overlapping	Yes Overlapping	Yes Overlapping
Implementation time	long	Less	Less	Depends on project
Flexibility	Rigid	Less Flexible	Highly Flexible	Flexible
Changes Incorporated	Difficult	Easy	Easy	Easy
Expertise Required	High	High	Medium	High
Cost Control	Yes	No	No	Yes
Resource Control	Yes	Yes	No	Yes

3.PROPOSED MODEL

Agent-Oriented Approaches – Gaia and ROADMAP

- Knowledge engineering software development methodologies are perceived (by some) as lacking because specifically for agent-based systems and this shortcomings are only partly addressed by extensions such as those seen for MAS-CommonKADS

- Gaia is a methodology for agent-oriented analysis and design supporting macro (societal) level as well as micro (agent) level aspects

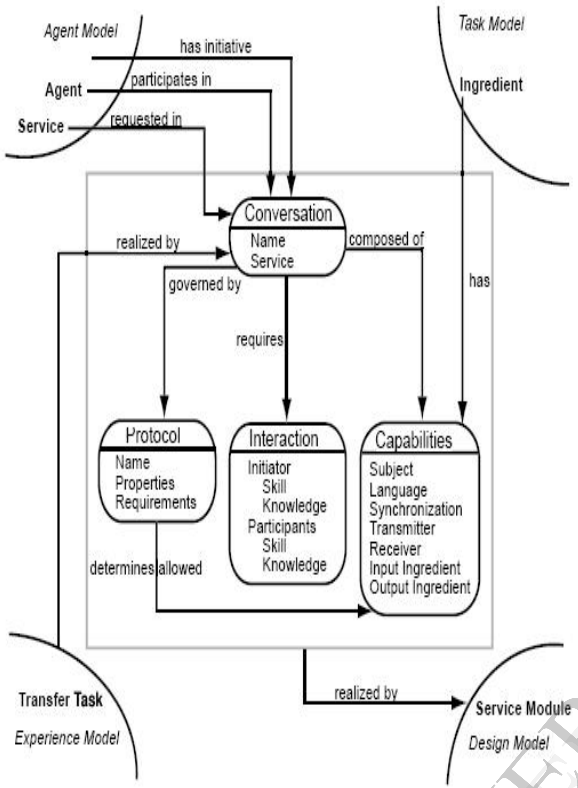

Figure1. Coordination Model

Agent-Oriented Approaches – SODA (Societies in Open and Distributed Agent spaces)

- SODA takes the agent environment into account and provides mechanisms for specific abstractions and procedures for the design of agent infrastructures
- *Agent societies* - exhibiting global behaviors not deducible from the behavior of individual agents
- *Agent environments* - the space in which agents operate and interact, such as open, distributed, decentralized, heterogeneous, dynamic, and unpredictable environments

But, *intra*-agent aspects are *not* covered – SODA is not a complete methodology; rather, its goal are to define a coherent conceptual framework, and a comprehensive software engineering procedure that accounts for the analysis and design of individual agents from a behavioral point of view, agent societies, and agent environments.

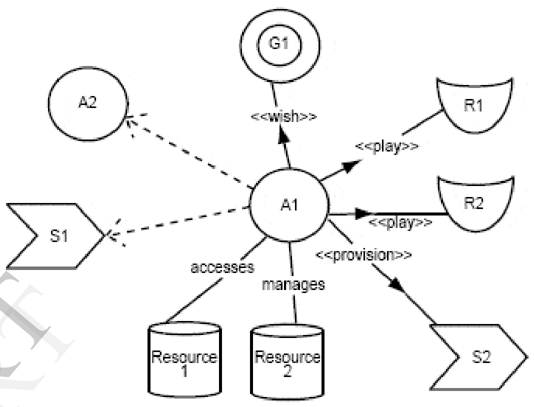

Figure2. Agent Diagram

Agent Communication Channel

- These components fit together to enable message transport between agents in two different ways
- FIPA also allows agents to interact in an arbitrary direct way but this does not require standards

FIPA defines external transport interfaces of agent platforms (via agent communication channel - ACC) in terms of overall transport architecture, MTPs and message envelopes the resulting external interfaces can then be accessed by either ACCs (method 1) or agents (method 2) directly from other platforms or on the same platform the internal interface of an agent to its platform's messaging services is not defined, to provide flexibility for toolkit developers.

A starting point: the OPEN Process Framework

We now briefly describe a suitable, existing process infrastructure upon which to build a facility to support the design of multi-agent systems. OO methodologies that are highly prescriptive and overly specified are hard to extend when a new variant or a new paradigm appears. What is required is a more flexible approach to building methodologies or processes. One such will be described here: OPEN, process components are selected from a repository and the actual methodology (or process) is constructed using identified construction and tailoring guidelines. OPEN thus provides a useful starting point because it is not only defined at the meta model level but is also itself componentized. Thus, adding further support for the design of intelligent agents is feasible, such potential extensions having been *a priori* designed into the meta-model architecture of the OPEN Process Framework (OPF).

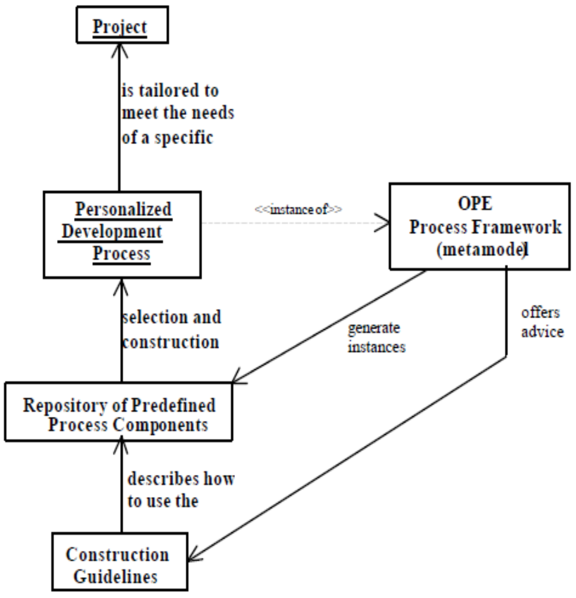
Project
is tailored to meet the needs of a specific
Personalized Development Process
<<instance of>>
OPE Process Framework (metamodel
selection and construction
generate instances
offers advice
Repository of Predefined Process Components
describes how to use the
Construction Guidelines

Figure 3. Creating a personalized development process

The OPF is a process meta-model or framework from which can be generated an organizationally- specific process (instance). Some of the major elements in this meta-model are Work Units (Activities, Tasks and Techniques wherein Activities and Tasks say “what” is to be done and Techniques say “how” it will be accomplished), Work Products and Producers3. Together, Work Units and Producers create Work Products and the whole process is structured temporally by the use of Stages (phases, cycles etc.). Each process instance is created by choosing specific instances of Activities, Tasks, Techniques etc. from the OPF Repository and specific configurations thereof (created by the application of the Construction Guidelines). OPEN thus provides a high degree of flexibility to the user organization.

Initial work in identifying new process components (e.g. Activities, Tasks, Techniques, Roles, Work Products) has already started. For example, in a newly proposed Task named *Identify intelligent agents* we suggest that identification of agents is in some ways an extension of ‘finding the objects’, although there are several clear differences (e,g, Odell, 2000). Agents are autonomous entities that have many similarities to objects. A major difference is that whereas an object receiving a request for service must deliver that service, an agent is empowered to say ‘no’. Agents act when “they feel like it”, and not necessarily when they receive a communication or other stimulus. Agents play roles with responsibilities. These responsibilities are not only equivalent to those for objects (responsibilities for doing, knowing and enforcing) but also towards achieving organizational goals (Jennings, 2001). However, they attempt to more closely mimic the behaviour of people and their decision making strategies than can objects. Consequently, there is a greater emphasis on the *roles* that are played by agents. Each role is defined by four attributes: responsibilities, permissions, motivations and protocols. Roles are already well supported in OPEN but need to be extended and refined to support the more sophisticated notion of roles in agent technology.

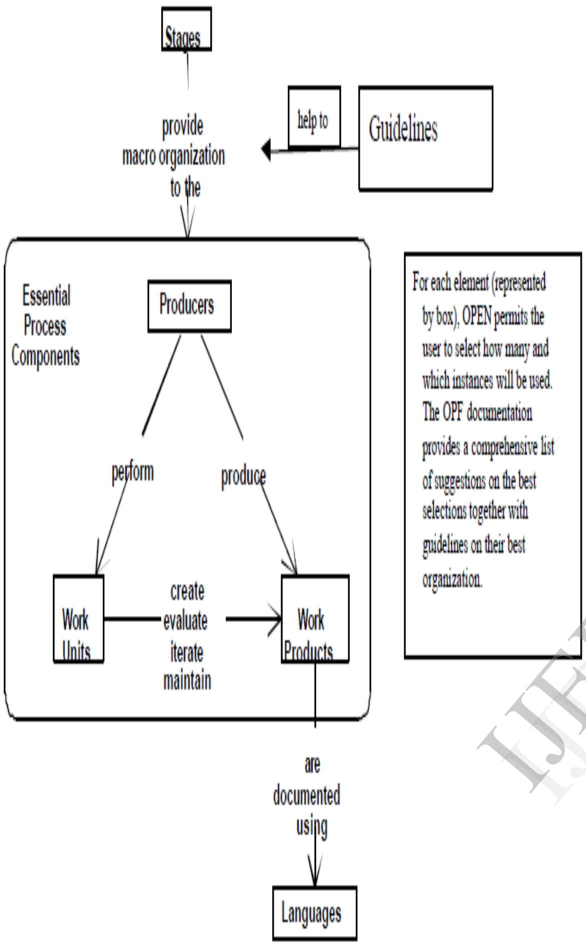
Stages
provide macro organization to the
help to
Guidelines
Essential Process Components
Producers
perform
produce
Work Units
create evaluate iterate maintain
Work Products
For each element (represented by box), OPEN permits the user to select how many and which instances will be used. The OPF documentation provides a comprehensive list of suggestions on the best selections together with guidelines on their best organization.
are documented using
Languages

Figure 3 The major elements of the OPF meta-model

4.EXPERIMENTAL ANALYSIS

Model competition from a consumer agent's perspective, let m be the number of consumers and n be the number of Agents. Since each consumer agent has m _ 1 competitors, let m' denotes m _ 1.

There are three thing which has been observed such as utility, success rate & negotiation period. Let us see the determination as:

1. *Utility*. The utility of a broker agent (U_{BA}) is determined as follows:

$$U_{BA} = \begin{cases} \frac{1}{n}\sum_{i=1}^{n}\left(U_{BA}^{i}(P^{i}) - \Gamma^{i}\right), & \\ \qquad \text{if all Cloud resources are obtained,} & \\ 0, \qquad \text{otherwise,} & \end{cases}$$

where U_{BA}^{i} is the utility of a broker agent obtained by establishing a contract for cloud resource R_i at the price of P^i, and Γ^i is the total amount of penalty fee that the broker agent should pay for R_i.

2. *Success rate*. A concurrent negotiation is considered successful if a broker agent can successfully negotiate for *all* of n types of cloud resources. Otherwise, the concurrent negotiation is considered unsuccessful.

3. *Negotiation speed*. The negotiation speed of a broker agent is determined as follows:

$$S_{BA} = t_{BA}/\tau_{BA},$$

5.PERFORMANCE MEASURE

Hence, $C(m,n) = 1 - [(m-1)/m]^n$ can be rewritten as

$$C(m,n) = 1 - (m'/(m'+1))^n.$$

It can be shown that $C(m,n) \to 1$ as $m'/n \to 0$ as follows:

$$\lim_{\frac{m'}{n}\to 0} C(m,n) = 1 - \left(\frac{m'}{m'+1}\right)^n = 1 - \left(\frac{\frac{m'}{n}n}{m'+1}\right)^n = 1.$$

Similarly, it can also be shown that $C(m,n) \to 0$ as $m'/n \to \infty$ as follows:

$$\lim_{\frac{m'}{n}\to \infty} C(m,n) = 1 - \left(\frac{m'}{m'+1}\right)^n = 1 - \left(1\Big/\left(1+\frac{1}{m'}\right)^{m'}\right)^{\frac{n}{m'}}.$$

Since $\lim_{m'\to\infty}(1+\frac{1}{m'})^{m'} = e$, $(1+\frac{1}{m'})^{m'}$ is finite. As $m'/n \to \infty$ it follows that $n/m' \to 0$. Therefore,

$$\lim_{\frac{m'}{n}\to \infty} C(m,n) = 1 - \left(1\Big/\left(1+\frac{1}{m'}\right)^{m'}\right)^{0} = 1 - 1 = 0.$$

6.CONCLUSION

There is as yet no single methodological approach that fits all purposes unsurprising given the breadth and scope of agent research and applications. Because of these challenges one approach being considered is the introduction of a meta-methodology that supports the various types of models described above and provides adequate mappings. An important prerequisite to bringing agent technology to market successfully is the availability of expressive and usable development tools, to enable software engineers to construct methodologies, define the various models listed above, and to achieve automatic model transformation as far as possible.

Finally, it appears that (independent of the methodology used) the question of how agent-based approaches can be embedded and migrated into mainstream IT infrastructures and solutions is another key factor to determine which elements of the current work on agent-oriented software engineering will be successful in real-world software engineering environments.

REFERENCES

[1].Agent-based Software Development Methodologies Brian Henderson-Sellers, University of Technology, Sydney (Australia) and Ian Gorton, Pacific Northwest National Laboratory (USA).

[2].Bresciani, P. and Giorgini, P. (2002). The TROPOS analysis process as graph transformation system, in Proceedings of the OOPSLA 2002, Odell, COTAR, Sydney, 1-12.

[3].Bresciani, P., Giorgini, P., Giunchiglia, F., Mylopoulos, J. and Perini, A., 2003, Tropos: an agent-oriented software development methodology, J. Autonomous and Multi-Agents (in press)

[4].Cabri, G., Leonardi, L. and ambonelli, F. (2002). Modeling role-based interactions for agents, in Proceedings of the OOPSLA 2002 Workshop on Agent-Oriented Methodologies (eds. J. Debenham, B. Henderson-Sellers, N. Jennings, and J. Odell), COTAR, Sydney, 13-20

[5].Debenham, J.K. and Henderson-Sellers, B. (2002). Full lifecycle methodologies for agent-oriented systems – the extended OPEN Process Framework. In Proceedings Agent-Oriented Information Systems (eds. P. Giorgini, Y. Lesperance, G. Wagner and E. Yu), Toronto, 87-101.

[6].Debenham, J.K. and Henderson-Sellers, B. (2003). Designing agent-based process systems –extending the OPEN process framework, chapter in Intelligent Agent Software Engineering, Idea Group Publishing, 160-190

[7].Debenham, J., Henderson-Sellers, B., Jennings, N. and Odell, J. (2002). Proceedings of the OOPSLA 2002 Workshop on Agent-Oriented Methodologies, COTAR, Sydney,
130pp, ISBN 0-9581915-6
Firesmith, D.G. and Henderson-Sellers, B. (2002).