Automated Software Development Methodology: An agent oriented approach

Sudipta Acharya*, Prajna Devi Upadhyay* and Animesh Dutta*

Abstract

In this paper, we propose an automated software development methodology. The methodology is conceptualized with the notion of agents, which are autonomous goal-driven software entities. They coordinate and cooperate with each other, like humans in a society to achieve some goals by performing a set of tasks in the system. Initially, the requirements of the newly proposed system are captured from stakeholders which are then analyzed in goal oriented model. Finally, the requirements are specified in the form of goal graph, which is the input to the automated system. Then this automated system generates MAS (Multi Agent System) architecture and coordination of the agent society to satisfy the set of requirements by consulting with the domain ontology of the system.

Keyword: Agent, Multi Agent System;Agent Oriented Software Engineering, Domain Ontology, MAS Architecture, MAS Coordination, Goal Graph.

1. Introduction

1.1 Agent and Multi agent system

An agent[1], [2] is a computer system or software that can act autonomously in any environment, makes its own decisions about what activities to do, when to do, what type of information should be communicated and to whom, and how to assimilate the information received. Multi-agent systems (MAS) [1], [2] are computational systems in which two or more agents interact or work together to perform a set of tasks or to satisfy a set of goals.

1.2 Agent Oriented Software Engineering

The advancement from assembly level programming to procedures and functions and finally to objects has taken place to model computing in a way we interpret the world. But there are inherent limitations in an object that makes it incapable of modeling a real world entity. It was for this reason that we move to agents and Multi agent systems, which model a real world entity in a better way. As agent technology has become more accepted, agent oriented software engineering (AOSE) also has become an important topic for software developers who wish to develop reliable and robust agent-based software systems [3], [4], [5]. Methodologies for AOSE attempt to provide a method for engineering practical multi agent systems. Recently, transformation systems based on formal models to support agent system synthesis are emerging fields of research. There are currently few AOSE methodologies for multi agent systems, and many of those are still under development.

2. Related Work

Recent work has focused on applying formal methods to develop a transformation system to support agent system synthesis. Formal transformation systems [6], [7], [8] provide automated support to system development, giving the designer increased confidence that the resulting system will operate correctly, despite its complexity. In [9] authors have proposed a Goal oriented language GRL and a scenarios oriented architectural notation UCM to help visualize the incremental refinement of architecture from initially abstract description. But the methodology proposed is informal and due to this the architecture will vary from developer to developer. In [10], [11] a methodology for multi agent system development based on goal model is proposed. Here, MADE (Multi Agent Development Environment) tool has been developed to reduce the gap between design and implementation. The tool takes the agent design as input and generates the code for implementation. The agent design has to be provided manually. Automation has not been shown for generation of design from user requirements. A procedure to map the requirements to agent architecture is proposed in [12]. The TROPOS methodology for building agent oriented software system is introduced in [13]. But the methodologies proposed in both [12] and [13] are informal approaches.

3. Scope Of Work

There are very few AOSE methodologies for automated design of the system from user requirements. But, most of the work follows an informal approach due to which the system design may not totally satisfy the user requirements. Also the system design varies from developer to developer. There is a need to reduce the gap between the requirements specification and agent design and to develop a standard methodology which can generate the design from user requirements irrespective of the developers. In this work we have concentrated on developing a standard methodology by which we can generate the design of software from user requirements which will be developer independent.

* *Department of Information technology, National Institute of Technology, Durgapur, India.*

In this paper, we develop an automated system which takes the user requirements as input and generates the MAS architecture and coordination with the help of domain knowledge. The basic requirements are analyzed in a goal oriented fashion [14] and represented in the form of goal graph while the domain knowledge is represented with the help of ontology [15]. The output of the developed system is MAS architecture which consists of a number of agents and their capabilities and MAS coordination represented through Task Petri Nets. The Task Petri Nets tool can model the coordination among the agents to maintain the inherent dependencies between the tasks.

4. Proposed Methodology

Figure 1 represents the architecture of our proposed automated system. The basic requirements are taken from the user as input. Since Requirements Analysis is an informal process, the input requirements can be captured from the user in the form of a text file or any other desirable format. These requirements are further analyzed and represented in the form of a Goal Graph. The domain knowledge is also an input and is represented in the form of ontology. The automated system returns the MAS Architecture and MAS Coordination as output. The MAS Coordination is represented in the form of Task Petri Nets. Thus, the automated system takes the requirements and the domain knowledge as input and generates the MAS Architecture and MAS Coordination as output. So, we can say,

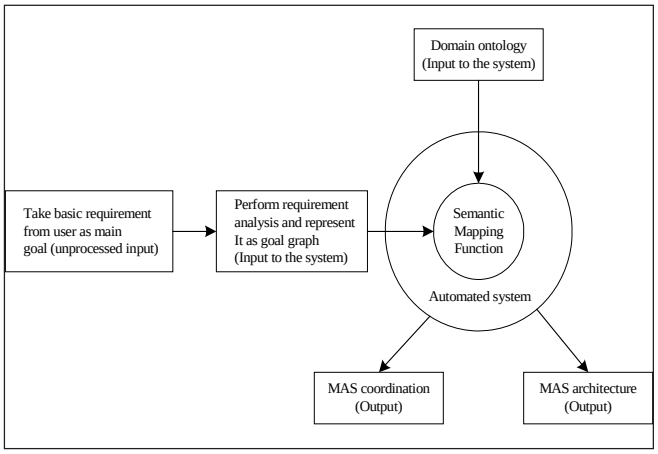

Figure 1. *Architecture of the proposed automated system*

MAS Architecture = f (Requirements, Domain ontology)

MAS Coordination = f (Requirements, Domain ontology, MAS architecture)

The architecture of the proposed automated system is described in the following sub-section.

4.1 Requirements represented by Goal Graph

Agents in MAS are goal oriented i. e. all agents perform collaboratively to achieve some goal. The concept of goal has been used in many areas of Computer Science for quite some time. In AI, goals have been used in planning to describe desirable states of the world since the 60s. More recently, goals have been used in Software Engineering to model requirements and non-functional requirements for a software system. Formally we can define Goal Graph as G = (V, E), consisting of

- A set of nodes $V = \{V_1, V_2,\ldots,V_n\}$ where each V_i is a goal to be achieved in a system, $1 \leq i \leq n$.
- A set of edges E. There are two types of edges, represented by ⟶ and ⟶◇ .
- A function, subgoal : (VxV) → Bool. subgoal (V_i, V_j) = true if V_j is an immediate sub goal of V_i.
- Function hb: (VxV) → Bool. $hb(V_i, V_j)$ = true if the user specifies that goal represented by V_i should be satisfied before the goal represented by V_j is satisfied.
- An ⟶◇ edge exists between two vertices V_i and V_j if subgoal (V_i,V_j) = true, $V_i, V_j \in V$.
- An ⟶ edge exists between two vertices V_i and V_j if hb (V_i,V_j) = true, $V_i, V_j \in V$.

4.2 Domain Knowledge represented by Ontology

A domain ontology [15] defined on a domain M is a tuple $O = (T_{CO}, T_{RO}, I, conf, <, B, IR)$, where we have extended the tuple definition by adding another function IR as per our requirements.

- $T_{CO} = \{c_1, c_2,\ldots,c_a\}$, is the set of concept types defined in domain M. Here, T_{CO} = {task, goal}. In diagram, a concept is represented by ▭
- T_{RO} = {consists_of, happened_before}. In diagram, a relation is represented by ⬭
- I is the set of instances of T_{CO}, from the domain M.
- conf: I → T_{CO}, it associates each instance to a unique concept type.
- $\leq : (T_C \times T_C) \cup (T_R \times T_R)$ → {true, false}, <(c,d) = true indicates c is a subtype of d
- B: T_R -> $\wp(T_C)$ where $\forall\ T_R$ B(r) = {c1,..,cn}, where n is a variable associated with r. The set {c1,..,cn} is an ordered set and could contain duplicate elements. Each ci is called an argument of r. The number of elements of the set is called the arity (or valence) of the relation. B(consists) = {goal,…,goal, task,…,task}
- IR: T_{RO} → $\wp$(I), where $\forall\ a_i \in \wp(I)$, if a_i is the ith element of $\wp$(I), then conf(a_i) = t_i, where t_i is the ith element of B(T_{RO}).

4.3 Semantic Mapping from requirements to Ontology Concepts

The process by which the basic keywords of the leaf node sub goal are mapped into concepts in the Ontology is called Semantic Mapping. In this paper, the aim of semantic mapping is to find out tasks from Domain Ontology, required to be performed to achieve a sub-goal given as input from the Goal Graph. Let there be a set of task concepts $T = \{t_1,t_2...t_n\}$ associated with a consists_of relation in an ontology. Let there be set of goal concepts $G = \{g_1,g_2...g_m\}$ also associated with that consists_of relation. Now let from user side requirements come of which after Requirements Analysis goal graph consists set of leaf node sub

goals, $G_0 = \{G_1, G_2, G_3, ...G_p\}$. Let ky be a function that maps a sub-goal to its set of keywords. The set of keywords for sub goal $G_i \bullet G_0$ can be represented by $ky(G_i) = \{ky_1, ky_2, ...ky_d\}$. Now the set of tasks T will be performed to achieve sub goal G_i iff either the mapping

$f : ky(G_i) \rightarrow G$ is a bijective mapping where $G_i \bullet G_0$, or there exists a subset of G_O, $\{G_i, G_j, \ldots, G_k\} \subseteq G_O$, such that

$f : ky(G_i) \cup ky(G_j) \cup ....\cup ky(G_k) \rightarrow G$ is a bijective mapping.

4.4 MAS Architecture

MAS architecture consists of set of agents with their capability sets i.e set of tasks that an agent can perform. Formally we can define agent architecture as,

< AgentID, {capability set}>

where AgentID is unique identification number of agent, and capability set is set of tasks $\{t_1, t_2, ..., t_n\}$ that the corresponding agent is able to perform. MAS architecture can be defined as a set of agents with their corresponding architectures.

4.5 MAS Coordination represented by Task Petri Nets

A Task Petri Nets is an extended Petri Nets tool that can model the MAS coordination. It is a six tuple, TPN = (P, TR, I, O, TOK, F_n) where

- P is a finite set of places. There are 8 types of places, $P = Pt \cup Ph \cup Pc \cup Pe \cup Pf \cup Pr \cup Pa \cup Pd$. Places Ph, Pc, Pe, Pf exist for each task already identified by the interface agent. The description of the different types of places is:

1. P_h: A token in this place indicates that the task represented by this place can run, i.e. all the tasks that were required to be completed for this task to run are completed.
2. P_c: A token in this place indicates that an agent has been assigned for this task.
3. P_e: A token in this place indicates agent and resources have been allocated for the task represented by the place and the task is under execution by the allocated agent.
4. P_f: A token in this place indicates that the task represented by this place has finished execution.
5. P_r: such a place exists for each type of a resource in the system , $\forall\, r_i \exists\, P_{ri}\, r_i \in R, 1 \le i \le q$
6. P_a: such a place exists for each instance of an agent in the system , $\forall\, a_i \exists\, P_{ai}\, a_i \in A, 1 \le i \le p$
7. P_t: it is the place where the tasks identified by the interface agent initially reside.
8. P_d: such place is created dynamically after the agent has been assigned for the task and the agent decides to divide the tasks into subtasks. For each subtask, a new place is created.

- TR is the set of transitions. There are 5 types of transitions $TR = th \cup tc \cup te \cup tf \cup td$, where th , te , tf exist for every task identified by the interface agent.

1. t_h: This transition fires if the task it represents is enabled i.e. all the tasks which should be completed for the task to start are complete.
2. t_c: This transition fires if the task it represents is assigned an agent which is capable of performing it.
3. t_e: This transition fires if the all resources required by the task it represents are allocated to it.
4. t_f: This transition fires if the task represented by the transition is complete.
5. t_d: This transition is dynamically created when the agent assigned for the task it represents decides to split the task further into sub-tasks. The subnet that is formed dynamically consists of places and transitions all of which are categorized as P_d or t_d respectively.

- I is the set of input arcs, which are of the following types

1. $I_1 = P_t X\, t_h$: task checked for dependency
2. $I_2 = P_r X\, t_e$: request for resources
3. $I_3 = P_e X\, t_f$: task completed
4. $I_4 = P_f X\, t_h$: interrupt to successor task
5. $I_5 = P_c X t_d \cup P_a X t_d \cup P_r X t_d \cup P_d X t_f$: input arcs of the subnet formed dynamically

- O is the set of output arcs, which are of the following types:

1. O_1: $t_h X P_h$: task not dependent on any other task
2. O_2: $t_c X P_c$: agent assigned
3. O_3: $t_e X P_e$: resource allocated
4. O_4: $t_f X P_r$: resource released
5. O_5: $t_f X P_f$: Task completed by agent
6. O_6: $t_f X P_a$: agent released
7. O_7: $t_d X P_d$: output arcs of the subnet formed dynamically

- TOK is the set of color tokens present in the system, $TOK = \{TOK_1, TOK_2, \ldots, TOK_X\}$, where each TOK_i, $1 \le i \le x$, is associated with a function assi_tok defined as:

assi_tok: TOK → Category X Type X N, where, Category = set of all categories of tokens in the system= {T, R, A}, Type = set of all types of each $category_i$ " Category i.e. Type= $T \cup R \cup A$, N is the set of natural numbers. Let $assi\_tok(TOK_i) = (category_i, type_i, n_i)$. The function assi_tok satisfies the following constraint:

$\forall\, TOK_i\, (category_i = R) \rightarrow \{(type_i \in R) \wedge (1 \le n_i \le inst\_R(type_i))\}$

$\forall\, TOK_i\, (category_i = A) \rightarrow \{(type_i \in A) \wedge (1 \le n_i \le inst\_A(type_i))\}$

$\forall\, TOK_i\, (category_i = T) \rightarrow \{(type_i \in T) \wedge (n_i = 1)\}$

assi_tok defines the category, type and number of instances of each token.

- F_n is a function associated with each place and token. It is defined as:

F_n : P X TO → ℘(TIME X TIME). For a token $TOK_k \in TOK$, $1 \le k \le x$, and place $P_l \in P$, $F_n(P_l, TOK_k) = \{(a_i, a_j)\}$, a_i is the entry time of TOK_k to place P_l and a_j is the exit time of TOK_k from place P_l. For a token entering and exiting a place multiple times, $|F_n(P_l,TOK_k)|$ = number of times TOK_k entered the place P_l.

The process by which MAS architecture and MAS coordination is generated from requirements is shown as a flowchart in Figure 2.

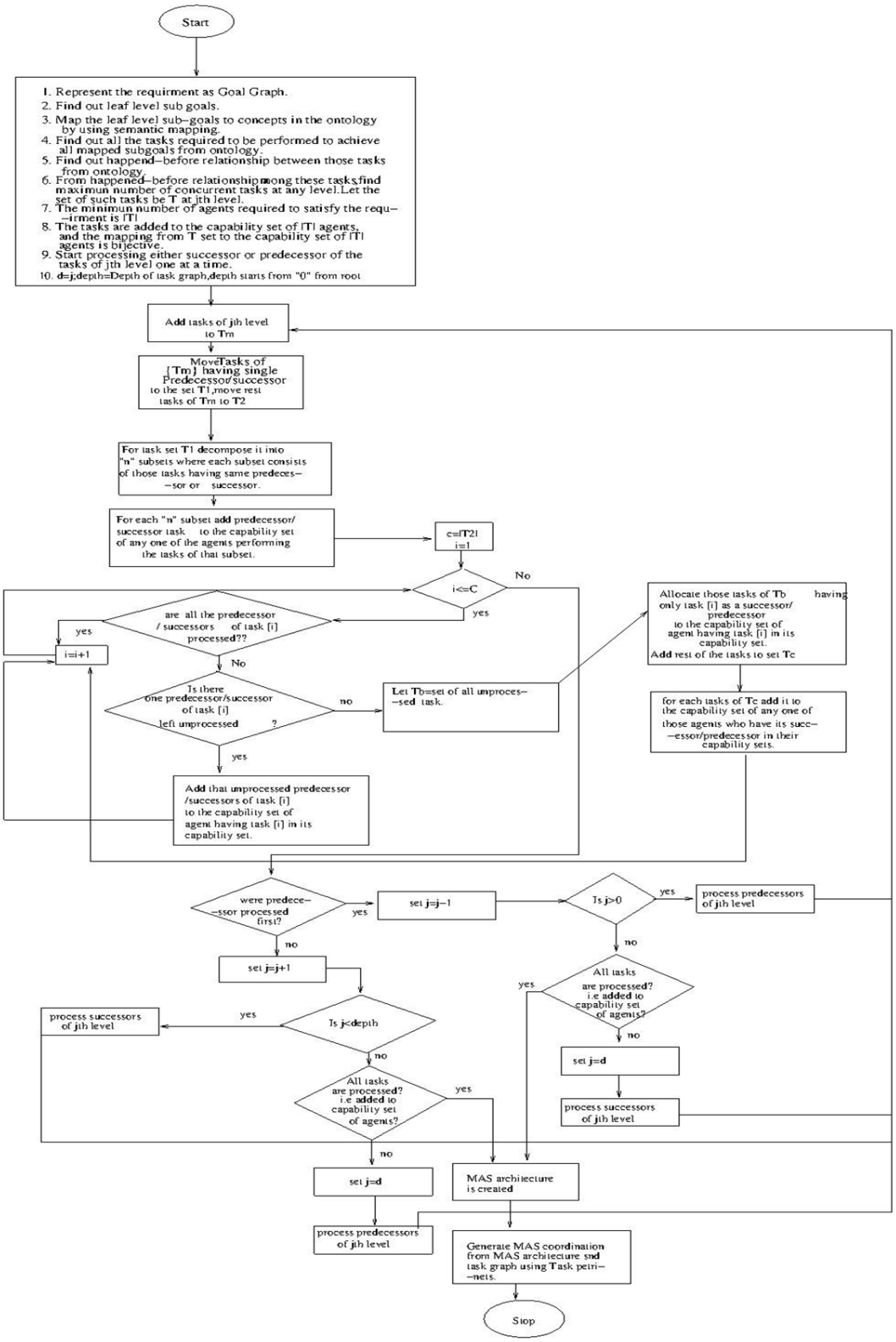

Figure 2. *Flowchart of the proposed methodology*

5. Case Study

Let us start with the case study by applying our proposed methodology. We take "Library system" as our case study application. Figure 3 shows the ontology of a Library System. The ontology consists of some concepts and relations. There is a TASK concept type in the ontology which describes the task that should be performed to achieve some goal. There are other concepts that collectively describe some sub-goal to be achieved in the library system. For e.g. the concept types "Check", "Validity", "Member", collectively describe the sub-goal "Check the membership validity". There are two types of relations i) consists_of and ii) happened_before. Here we denote "happened_before" relationship by "H-B". The "consists_of" relation exists between some set of concepts describing a sub-goal and a set of instances of TASK concept. The "happened_before" relation exists between two instances of TASK concept. In the figure, the "consists_of" relation has incoming arcs from the concepts types "Check", "Validity, "Member" and outgoing arcs to TASK concepts "Get library identity card of member", "Check for validity of that ID card". It means that these two tasks have to be performed to achieve sub-goal described by concepts "Check", "Validity", "Member" i.e. tasks "Get library identity card of member" and "Check for validity of that ID card" have to be performed to achieve sub-goal "Check the membership validity". The happened_before relationship exists between these two tasks which means that task "Check for validity of that ID card" cannot start until task "Get library identity card of member" is completed. Now consider from user side requirements come as "Delete account of member with member id <i> and book with book id <j> from database."It is the main goal.

Step 1: We have to perform goal oriented Requirements Analysis of main requirements. It is an informal process performed by the Requirements Analysts and after Requirements Analysis, it is represented by Goal Graph shown in Figure 4.

Step 2: The leaf node sub goals are given as input to the automated system. By semantic mapping [16], [17] system maps each basic keyword of leaf sub goals to the goal concepts of ontology, and finds out set of tasks required to be performed to achieve those sub-goals. This is shown in Figure 5.

Step 3: The tasks that we get from step 2 are used to form task graph. Dependency between these tasks is known from ontology. The task graph is shown in Figure 6 where task A implies "Check that requirement of book id <j> < threshold, if yes then continue, else stop.

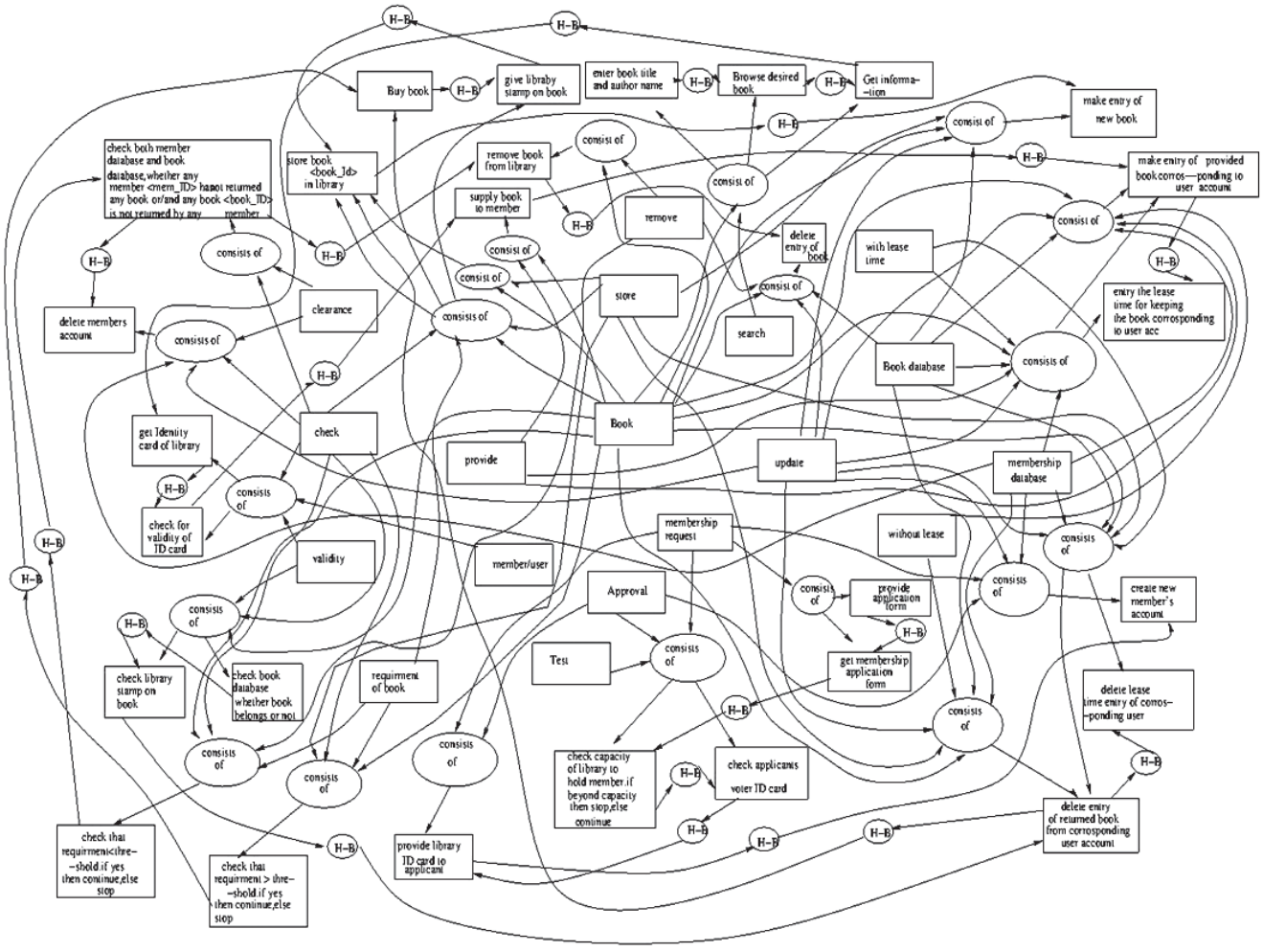

Figure 3. *Ontology Diagram of Library System*

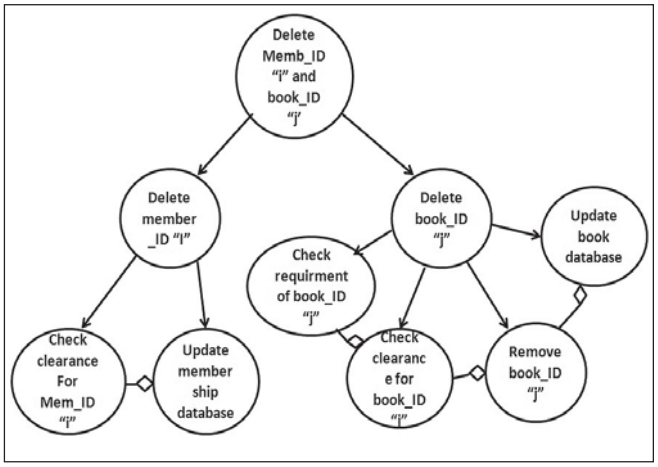

Figure 4. *Goal Graph representation of basic requirements*

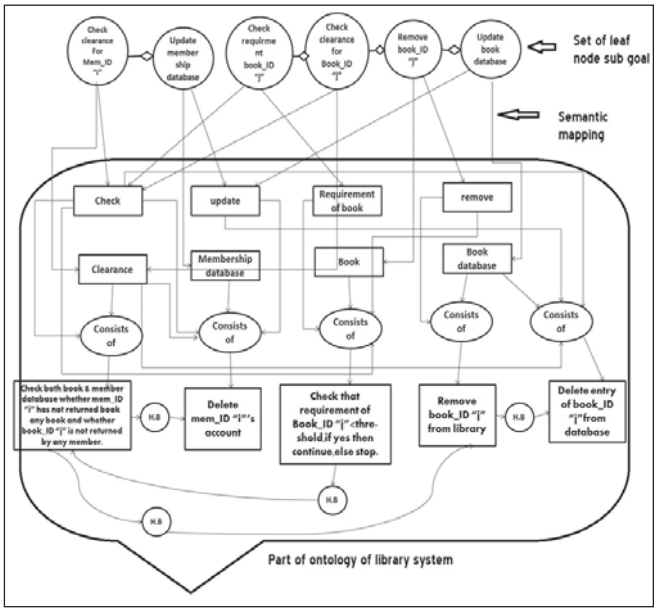

Figure 5. *Procedure for Semanic Mapping*

Task B implies "Check both book & member database whether member id <i> has not returned book, and any book <j> is not returned by any member."
i. e., in task B two checking operations are there.
Task C implies "Delete member id <i> account."
Task D implies "Remove book id <j> from library"
Task E implies "Delete entry of book id <j> from database"

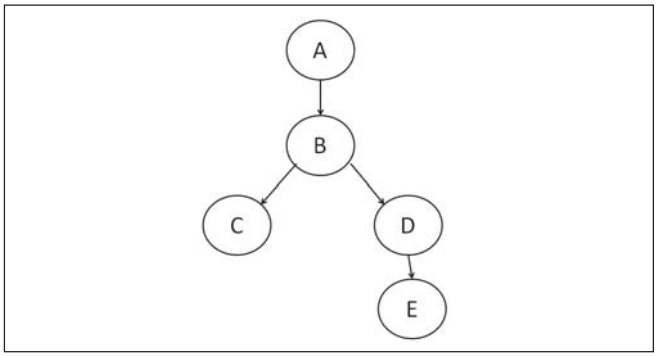

Figure 6. *Task Graph for the set of tasks found from Semantic Mapping*

Step 4: Using Task Graph of Figure 6, we find out the number of agents and their capability set following the methodology shown as a flowchart in Figure 2. The maximum number of concurrent agents at any level is 2, so we create 2 agents- A_1 and A_2. Let C be assigned to A_1's capability set and D to A_2's capability set, $<A_1, \{C\}>$, $<A_2, \{D\}>$. Both C and D have single predecessor, B. So, B is added to the capability set of either A_1 or A_2. Let it be added to the capability set of A_1. So, we have $<A_1, \{B, C\}>$. Now, B has a single predecessor, A. So, A is added to the capability set of A_1. So, we have $<A_1, \{A, B, C\}>$. There are no other predecessors at level higher than A. D has a single successor, E. So, E is added to the capability set of A_2. So, we have $<A_2, \{D, E\}>$. The total number of agents deployed is 2 and the MAS architecture is $<A_1, \{A, B, C\}>$, $<A_2, \{D, E\}>$.

Step 5: Using the Task Graph of Figure 6 and MAS architecture developed in step 4, MAS coordination is formed i.e. to satisfy user requirements, how a set of required agents (A_1, A_2) will perform a set of required tasks (A, B, C, D, E) collaboratively can be represented by Task Petri Nets shown in Figure 7.

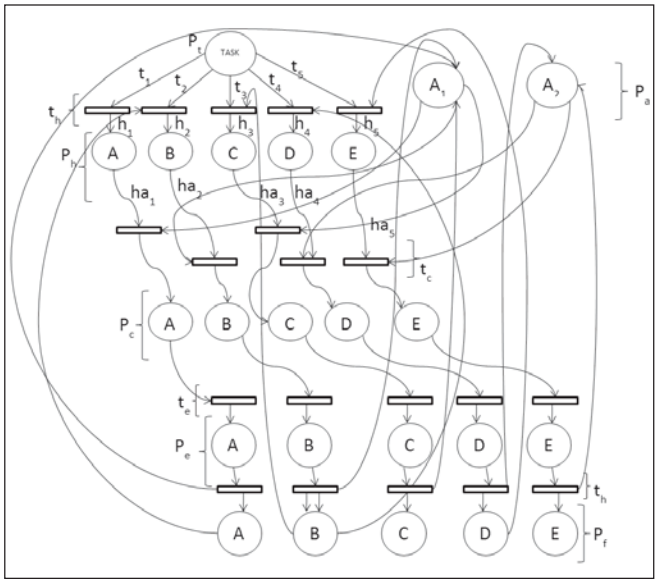

Figure 7. *Task Petri Nets representation of MAS coordination*

6. Conclusion

In this paper, we have developed an automated system to generate MAS architecture and coordination from the user requirements and domain knowledge. It is a formal methodology which is developer independent i.e. it produces same MAS architecture and coordination for the same set of requirements and domain knowledge. The future work is to include a verification module to check whether the developed architecture satisfies the requirements. The module can work at two levels, firstly after Requirements Analysis, it can check whether the analysis satisfies main requirements, and secondly, it can verify whether the MAS coordination satisfies main requirements.

7. References

[1] G. Weiss, Ed., *Multiagent systems: a modern approach to distributed artificial intelligence*, MIT Press (1999).

[2] M. J. Wooldridge, *Introduction to Multiagent Systems*, John Wiley & Sons, Inc(2001).

[3] N. Jennings, "On Agent-based Software Engineering," *Artificial Intelligence*, Vol.117 (2000), pp. 277-296.

[4] J. Lind, "Issues in Agent-Oriented Software Engineering," In P. Ciancarini , M. Wooldridge (eds.), *Agent-Oriented Software Engineering: First International Workshop, AOSE 2000*. Lecture Notes in Artificial Intelligence, Vol. 1957, Springer-Verlag, Berlin.

[5] M. Wooldridge, , P. Ciancarini, "Agent-Oriented Software Engineering: the State of the Art," In P. Ciancarini, M. Wooldridge, (eds.), *Agent-Oriented Software Engineering: First International Workshop, AOSE 2000*. Lecture Notes in Artificial Intelligence, Vol. 1957, Springer-Verlag, Berlin Heidelberg (2001), pp.1-28.

[6] C. Green, D. Luckham, R. Balzer , et al, "Report on a Knowledge-Based Software Assistant". In C. Rich, R. C. Waters, (eds.), *Readings in Artificial Intelligence and Software Engineering*. Morgan Kaufmann, San Mateo, California (1986), pp.377-428.

[7] T. C. Hartrum , R Graham, "The AFIT Wide Spectrum Object Modeling Environment: An AWESOME Beginning", *Proceedings of the National Aerospace and Electronics Conference*, pp.35-42, 2000.

[8] R. Balzer, T. E. Cheatham, Jr., and C. Green, "Software Technology in the 1990's: Using a new Paradigm," *Computer*, pp. 39-45, Nov 1983.

[9] L. Liu, E. Yu, "From Requirements to Architectural Design –Using Goals and Scenarios".

[10] Z. Shen, C. Miayo, R. Gay, D. Li, "Goal Oriented Methodology for Agent System Development", *IEICE Trans. Inf. & Syst.*, Vol.E89-D, No.4, April 2006.

[11] S. Zhiqi, "Goal oriented Modelling for Intelligent Agents and their Applications", Ph.D. Thesis, Nanyang Technological University, Singapore, 2003.

[12] Clint H. Sparkman, Scott A. DeLoach, Athie L. Self, "Automated Derivation of Complex Agent Architectures from Analysis Specifications", *Proceedings of the Second International Workshop On Agent-Oriented Software Engineering (AOSE-2001)*, Montreal, Canada, 2001.

[13] P. Bresciani, A. Perini, P. Giorgini, F. Giunchiglia, J. Mylopoulos, "Tropos: An Agent-Oriented Software Development Methodology," *Autonomous Agents and Multi-Agent Sytems*, Vol. 8, pp.203-236, 2004.

[14] Paolo Giorgini, John Mylopoulos, and Roberto Sebastiani, "Goal-Oriented Requirements Analysis and Reasoning in the Tropos Methodology," *Engineering Applications of Artificial Intelligence*,Vol. 18, pp.159-171, 2005.

[15] P.H.P. Nguyen, D. Corbett, "A basic mathematical framework for conceptual graphs," *IEEE Transactions on Knowledge and Data Engineering*, Vol.18, Iss. 2, 2005.

[16] Haruhiko Kaiya, Motoshi Saeki, "Using Domain Ontology as Domain Knowledge for Requirements Elicitation," 14th IEEE International Requirements Engineering Conference (RE'06), 2006.

[17] Masayuki Shibaoka, Haruhiko Kaiya, and Motoshi Saeki, "GOORE : Goal-Oriented and Ontology Driven Requirements Elicitation Method," J.-L. Hainaut et al. (eds.): ER Workshops 2007, LNCS 4802, pp. 225–234, 2007.