International Journal of Computer Applications (0975 – 8887)
Volume 169 – No.9, July 2017

User Oriented Requirements Engineering for Agent Oriented Systems

Anuja Soni
Deen Dayal Upadhyaya College
University of Delhi

ABSTRACT

Poor requirements are one of the principal reasons for failures of projects. A casual attitude to the user-requirements at the requirements stage leaves little room for improvement at the final stage of software development. This study is aimed to act as a bridge between the real world needs of users alleged as requirements and potential of developer to intensively investigate their needs for Agent Oriented Systems. This work employs the notion of the User Story Card (USC) for requirements elicitation that acts as a powerful tool to reflect the true requirements of users in the final artifact. In addition, this work presents Agent Cards(ACs) to define as well as validate the requirements to ensure that the requirements truly represent users' expectations so that the system based on these requirements eventually would lead to their satisfaction.

Keywords

User Story Card (USC); Agent Card (AC); Multi-Agent System (MAS), Requirements Engineering, Validation

1. INTRODUCTION

From the commencement of requirements gathering till its conclusive form, the developer is subject to face several vicissitudes. The lack of concreteness in the requirements leads to the devastation of projects. Therefore, for the successful accomplishment of the system, the developer is required to cautiously complete and validate the requirements before proceeding to design stage [1].

Requirements Engineering (RE) is a structured process of acquiring, defining, validating and specifying the requirements of a system. There are various approaches recognized by researchers in literature for requirements engineering such as Goal based requirements engineering [2], Agile based visualization techniques [7], Formal Tropos [1], Agent based requirements Engineering [4] etc. Out of these methodologies, Agent based requirements engineering models problems in terms of autonomous interactive component agents that is proving to be a more natural way of representing task allocation, team planning and user preferences. An agent based system provides a flexible mechanism to model the stakeholders [5] that facilitate the mutual influence between envisioned system and human context where it will work [6].

Agent Oriented requirements engineering addresses the requirements of a system in terms of agents. Software agents are computer programs that act autonomously on behalf of their users across open and distributed environments. Various requirements frameworks such as i* [1], ConGolog [1], REF [5] have been recommended for Agent Oriented systems. All these approaches have their own potencies and limitations. i* framework supports only early phase of requirements engineering (RE); ConGolog is expressive logic based formal framework that supports late phase of RE activities. Concerted form of i* and ConGolog framework [1] supports early to late phase of requirements engineering. These techniques assist the developers to elicit the requirements from the users; however users themselves are not involved to provide their requirements.

Non-specific requirements wander off the developers from the users' true needs and expectations resulting in defective requirements and the consequences of these defects are repulsive [4, 6] such as:

Likelihood of misinterpretation of requirements due to lack of users' participation:

-Unsatisfied users
-Cost and resource overrun
-A poor quality product
-Delayed system delivery
-Expensive maintenance
-Exhausted and demoralized software development team

In light of these issues, a User Oriented Requirements Engineering is required that necessitates the involvement of the users for obtaining their true requirements.

To reflect users' expectations into the final requirements document; this paper proposes a User Oriented Requirements Engineering methodology for Agent Oriented systems that assists developers in acquiring the requirements directly from the users and prevents the chances of misinterpretation among users and developers. This methodology facilitates the developers to refine the requirements iteratively to have a complete, consistent and precise list of requirements before proceeding to design stage. In this framework, the requirements are recorded in a set of USCs [7] developed jointly by customer representatives and the development team. USCs are mapped to ACs that is an easy and effective means to understand characteristics of an agent in terms of goals and tasks. This work argues that highly interactive USCs and goal oriented ACs; collectively result in a requirements artifact through a mapping process that is precise, consistent and comprehensive. Also, this approach ensures that user-requirements are truly reflected into the final requirements document as it is the user who ultimately decides the success of a system.

The organization of the paper is as follows: Section II presents the User Oriented Requirements Methodology for Agent Oriented systems. Section III exhibits the applicability of the proposed approach and finally section IV concludes the paper.

2. USER ORIENTED REQUIREMENTS METHODOLOGY FOR AGENT-ORIENTED SYSTEMS

Requirements Methodology involves the process of acquiring and then establishing the user oriented view of the requirements of a system.

Poorly collected requirements in the early phases of the software development can be exceedingly costly in the later stages of software production. This paper focuses on requirements methodology for Agent-Oriented system that comprises of the activities such as requirements elicitation, definition, validation and specification. The Fig. 1 illustrates the main steps of the methodology. The user stories are collected from the users using the interface USCs. USCs are expanded into a number of sub-USCs that facilitate in extracting complete set of user-requirements 'R'. Requirements 'R' are mapped to various ACs that eventually help in mapping USCs to ACs iteratively till all USCs are mapped to ACs. A reverse mapping from ACs to USCs is applied to locate unmapped ACs. Users are involved in workshops for discussion over unmapped requirements.

The process is repeated till all requirements are refined. Refined requirements are validated for their consistency and completeness and act as a base line document for requirements specification document. The same is described in detail in the subsequent sub sections.

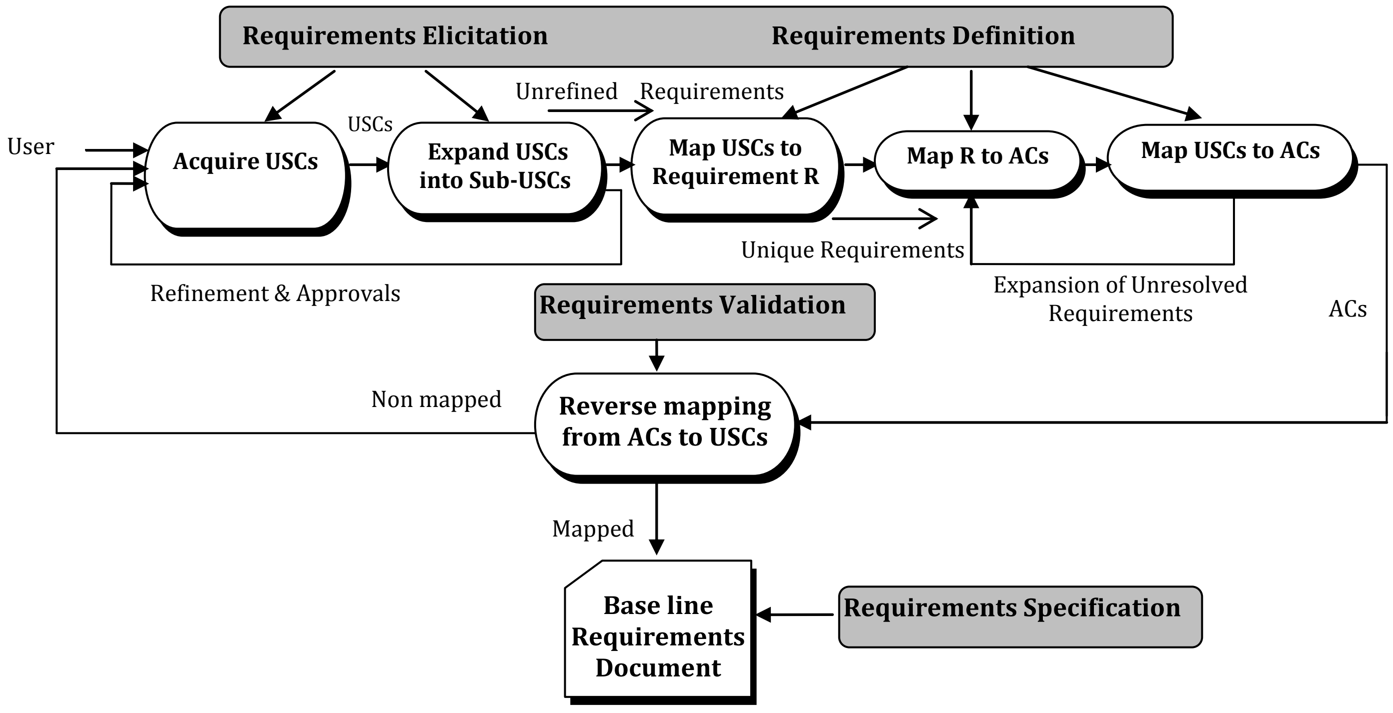


Fig 1: User-Oriented Requirements Engineering Methodology for Agent Oriented Systems

2.1 Requirements Elicitation

Requirements elicitation involves gathering requirements of a system from various stakeholders. Various techniques for requirements elicitation are mentioned in the literature namely interviews, observations, prototyping [3], textual-based artifacts like use cases [11], plain natural language [15], a hybrid approach (integration of user interaction and natural language processing) [12], Value Gap Model [14], Key words mapping based requirements elicitation [13], Pattern-based requirements elicitation [10] etc.

Elicitation strategies which produce requirements in the form of high level designs run the risk of creating requirements which are ambiguous to the user community.

These requirements may not be verifiable by the users because they do not adequately understand the design language. Also, requirements expressed as a design is much more likely to incorporate additional decisions not reflecting user needs i.e. requirements will not be precise and necessary.

Many requirements elicitation methods involve error prone recording process and delayed cost-estimation and thereby focus on fulfilling the requirements list rather than the intended user-goals [3]. The ever-increasing demand of high quality software has elevated the need for users' involvement in requirements elicitation. To enhance the users' involvement in the requirements elicitation, this work uses a simple and user-Oriented concept of User Story.

User Story assists the developers in acquiring requirements directly from users to reduce the likelihood of defects in the requirements [3]. User Stories are written using the prescribed template shown below that is disseminated among various stakeholders involved in requirements elicitation [9].

As a **< User >** I want to achieve **< Goal >** So That **< Reason** >

The User Stories signify goals of a system placed in short sentences in active voice. The <Goal> clause of the User Story signifies the actual requirements of users and is likely to be exhibited by the final system, whereas the term <User> signifies the role of the user in a real world environment.

The goals along with many other parameters like User Story Number, User Story Title, User Role, User Name, Date of Creation, Time-Estimation are recorded on USCs developed jointly by customer representatives and developers.

The <Reason> clause in the User Story facilitates the users to specify the motive for their needs. After obtaining the requirements of the users, the "reason" turns out to be self evident, therefore it is not recorded on the USC for further processing. The structure of USC is illustrated in Fig. 2. USC serves as a means to eliminate the gap between users and developers during requirements elicitation. These USCs can be further expanded and refined into a number of sub USCs after a thorough discussion with users.

The process of expansion of User Stories into sub-stories continues iteratively till all requirements are captured from users up to their satisfaction. USCs once validated form an artifact for Software Requirements Specification.

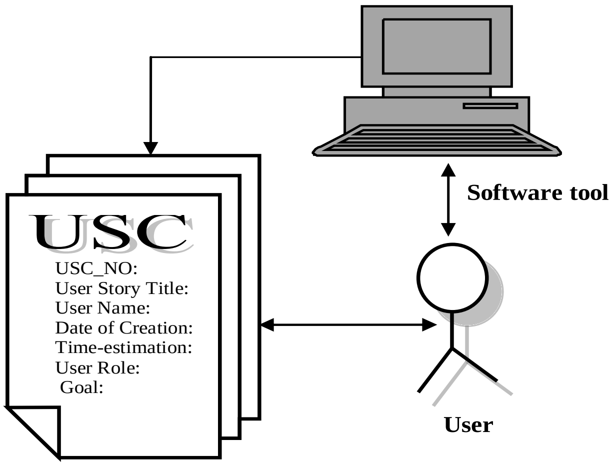
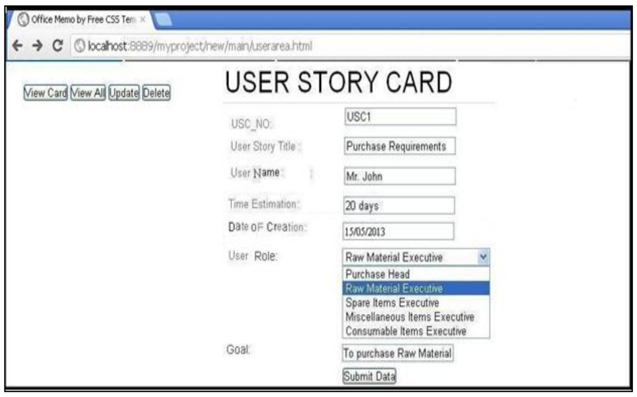


Fig 2: Structure of USC

An example of USC is illustrated in Fig. 3. These user stories can be stored manually or electronically. USCs can be effectively managed by using the tag interface tools that support quick access to user stories during the period of appraisal [9].


Office Memo by Free CSS Tem
localhost:8889/myproject/new/main/userarea.html
View Card
View All
Update
Delete
Submit Data

USER STORY CARD

USC_NO:	USC1
User Story Title:	Purchase Requirements
User Name:	Mr. John
Time Estimation:	20 days
Date oF Creation:	15/05/2013
User Role:	Raw Material Executive
Goal:	To purchase Raw Material

Purchase Head
Raw Material Executive
Spare Items Executive
Miscellaneous Items Executive
Consumable Items Executive

Fig 3: Example of USC

2.2 Requirements Definition

Requirements definition translates the requirements captured from the users into the services, that the final system is expected to provide. This work employs Agent-Oriented approach to define the requirements of a system in terms of agents.

Multi-Agent System (MAS) is a system composed of various agents namely User Interface Agent, Goal Oriented Agent, Monitoring Agent, Security Agent and Communicative Agent etc. that cooperate to solve a complex problem in a decentralized way and are defined as follows:

User Interface Agent: User Interface Agent is conscientious for user interface and input output facets.

Goal Oriented Agent: Goal Oriented Agent is responsible for the achievement of major goals of organization.

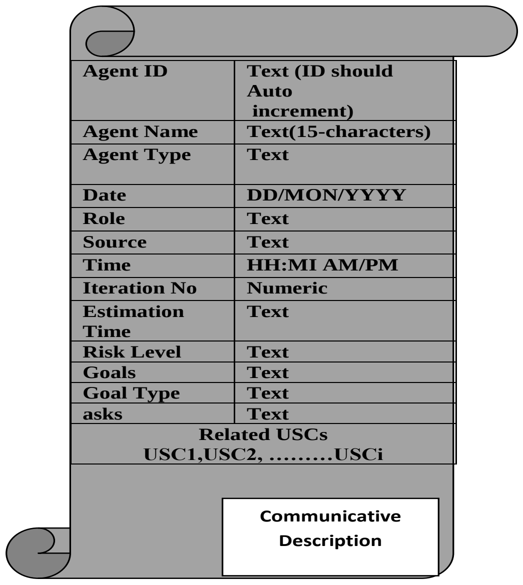
Monitoring Agent: Monitoring Agent is responsible for registering, de-registering agents and co-ordinate their activities.

Security Agent: Security Agent takes care of security features imposed on data confined with a single agent or shared by multiple agents.

Communicative Agent: Communicative Agent is responsible for facilitating communication among two or more agents.

This paper presents AC [16], as a means for defining the requirements in terms of the goals that are expected to be achieved by various agents in MAS. AC acts as a repository of the information in terms of goals, tasks and other associated parameters of various agents that facilitate the developer to foresee the requirements of a system in a broader way.

AC enables the developers to establish the comprehensive list of requirements of agents in terms of roles, goals, tasks and services. The template of AC is shown in Fig. 4 that works as a repository of the requirements for an agent [8].



Agent ID	**Text (ID should Auto increment)**
Agent Name	**Text(15-characters)**
Agent Type	**Text**
Date	**DD/MON/YYYY**
Role	**Text**
Source	**Text**
Time	**HH:MI AM/PM**
Iteration No	**Numeric**
Estimation Time	**Text**
Risk Level	**Text**
Goals	**Text**
Goal Type	**Text**
asks	**Text**
Related USCs USC1,USC2, ...USCi	

Communicative Description

Fig 4: Template of AC

```
Type AGENT_STRUCTURE is record
Agent_ID              : varchar2;
Agent_Name                         : varchar2;
Agent_Roles[]                      : ROLE;
Date                               : date;
Time                               : varchar2;
Source                             : varchar2;
Agent_Goals[ ]                     : GOAL;
Agent_Origin                       : ORIGIN;
End record;
Type GOAL is record
Goal_Name                          : varchar2;
Agent_Tasks [ ]                    : varchar2;
Type ROLE is record
Agent_Roles [ ]                    : varchar2;
Dependums[ ]                       : varchar2;
End record;
Type ORIGIN is record
USC_Name[ ]                         : varchar2;
USC_Id[ ]                           : varchar2;
End record;
```

Fig 5: Structure of AC

ACs are expanded form of USCs that leverage detailed understanding of requirements in the context of functionality of various agents. USCs incorporate simple requirements depicting user's perspective but, on the contrary ACs, designed by developers represent the broad perception of requirements. By fabricating ACs, developer reaches to additional goals and tasks and decides whether requirements already captured by users in the form of USCs are actually feasible to achieve. The structure of AC is shown in Fig. 5. The structure illustrates various agent attributes like Agent ID, Agent Name, Agent Type etc. that are explained in the following section.

Agent ID consists of agent identification number assigned by monitoring agent; Agent name incorporates the name of the associated agent; Agent-role indicates the behavior of an agent for accomplishing a goal and a goal for which one agent is dependent on another agent is called dependum. Origin defines the names and identification numbers of corresponding USCs from which the AC is generated.

The source indicates the personal details of the concerned persons who are accountable for allied USCs; date and time can have any standard format like dd/mon/yyyy and hours: minutes am/pm respectively depicting, when the AC was created; iteration number stipulates the iteration for agent implementation; estimation time is the faltering execution time for AC in terms of number of days; risk level refers to the risk (medium/high/low) associated with implementation of agent specified in AC; role indicates the role of the concerned agent; goals describe the main goal of the agent; tasks specify the sub-goals that agent would meet for achieving the main goal. Goal type can be hard goals, soft goals or maintenance goals [3]. Communicative description shows the dependencies among agents in terms of goal, task, resource and soft goal as prescribed by i* framework [2].

The algorithm to define the requirements consists of the following steps:

1) Formulate relation Ur : USCs→ R that maps user stories USCs to requirements R. This would result in reducing the ambiguity by tracing out redundant requirements.

Suppose

i. n→total no. of requirements
ii. i→no. of redundant requirements
iii. n-i→no. of unique requirements left in USCs that would be utilized in following step.

2) Determine relation A_r: R→ ACs that maps requirements 'R' obtained in previous step to one or more ACs.

3) Obtain the relation Ua: USCs→ACs from relations U_r and A_r that finally maps the USCs to the ACs.

The mapping process assists developer to explore additional requirements pertaining to agents and to obtain consolidated association of requirements with respect to USCs and ACs.

2.3 Requirements Validation

To achieve completeness and consistency of requirements procured, reverse mapping is done from ACs to USCs that traces out missing goals in USCs. Developer carries out reverse mapping to obtain a consistent, complete and unique list of requirements.

Assume

Task: T_m (m: no. of tasks linked with AC_r)
Goal: G_j (j: no. of Goals associated with AC_r)
USC: U_h (h: no. of USCs associated with AC_r)
r: number of ACs

$$\text{If } \forall\, T_m \in AC_r \;\; \exists G_j \text{ s.t. } \forall G_j \text{ is mapped to USC} \qquad (1)$$

Fig.6 ensures the completeness and consistency between AC and associated USCs if condition (1) turns out to be true else there can be two cases:

(i) Mapping of tasks to goals:

$$\text{If } \forall\, T_m \in AC_r \;\; \nexists G_j \in U_h \qquad (1.1)$$

Equation (1.1) implies that for every task there is not any existing goal. At this step, developer is required to assign some goals to newly captured tasks.

(ii) Mapping of goals to USCs:

$$\text{If } \forall\, T_m \in AC_r \;\; \exists G_j \text{ but } \nexists U_h \qquad (1.2)$$

Equation (1.2) implies that new goals have been seized in ACs. At this step, developer is required to conduct a workshop with users so that USCs can be updated according to these new additional goals.

The consistency and completeness in the requirements document can be ensured by the following procedure:

- Compare ∀ G_j of AC_r with all the goals of USCs.
- ∀ AC_r obtain the value of GE_r

GE_r is a one bit Goal Equality indicator that can be defined as:

$GE_r=1$ $\forall AC_r$:If $\forall G_j$ of AC_r is mapped to any of goals of U_h

$GE_r=0$ for any AC_r: If any G_j is not mapped to any of goals of U_h

In a similar manner, GE_r is calculated for all AC_r. For verifying consistency and completeness of the final requirements document, Equation (2) is used. Obtain Validation Factor (VF) by the following formula:

$$VF = \prod_{p=1}^{n} GE_p \quad \text{where n:no.of ACs} \qquad (2)$$

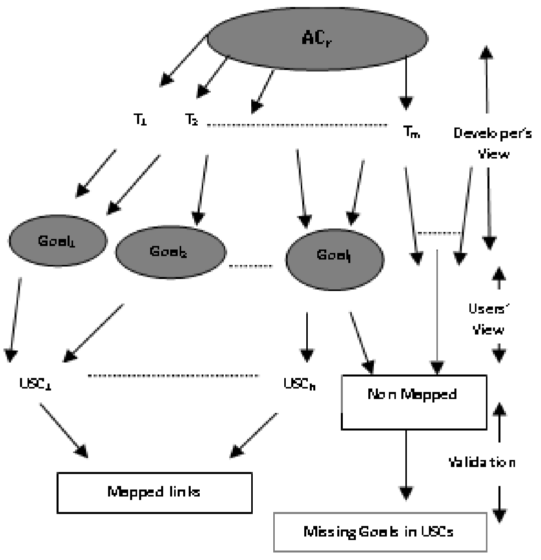


Fig 6: Validating requirements through reverse mapping

VF=1 will ensure condition (1) which implies that every task in all ACs and subsequently all goals are mapped to all USCs.

VF=0 will lead to condition (1.1) or (1.2) that implies that there is some inconsistency in the user-requirements obtained using USCs and goals accumulated in ACs. This indicates generation of additional goals and tasks during fabrication of

ACs. Developer cannot proceed further unless these goals and tasks are approved by users. Developer is required to conduct a workshop with users to take their consent before approaching to next phase of RE. This process is continued till the value of VF is obtained as 1. Value of VF as 1 ensures validation of all goals and tasks.

2.4 Requirements Specification

The requirements specification entails the complete behavior of the system that act as a contract between the system developers and users. The methodology proposes that when the value of VF comes out as 1 for all ACs; that is when all the goals of ACs are mapped to all USCs, then the USCs and their corresponding ACs act as a complete document for requirements specification.

The proposed approach bridges the gap between user and developer by facilitating ACs to capture additional goals/tasks and a reverse mapping is employed to ensure completeness and consistency in the final requirements document.

3. CASE STUDY

To see the application of the method, a case study of Material Management Multi Agent System (MM MAS) is performed. MM MAS is composed of User Interface Agent, Goal Oriented Agent, Monitoring Agent, Security Agent and Communicative Agent as explained in section II. This case study is aimed to focus on the requirements of different agents captured in the form of USCs, propagated in the form of ACs and validated through a reverse mapping process. MM MAS comprises following activities:

--Material planning and purchasing
--Inventory control
--Receiving and accounting
--Store keeping
--Disposal of surplus store

Table 1. Various Agent Types in MMMAS Iteration Wise

Iterations		**I**	**II**	**III**	**IV**
Agent Type	***Agent Role***				
User Interface Agent	Material Interface Agent	■	©	©	©
Goal Oriented Agents	Indent Receiver		■	©	©
	Inventory Analyst	■	©	©	©
	Inventory Controller	■	©	©	©
	Store Receiver Agent	■	©	©	©
	Disposal Agent			■	
Monitoring Agent	Material Monitoring Agent	■	©	©	©
Security Agent	Material Security Agent	■	©	©	©
Communicative Agents	Purchase Agent		■	©	©
	Store Agent		■	©	©
	Surplus Agent			■	©

■ indicates agents to be handled for recent iteration
© indicates changes to be handled in subsequent iterations.

TABLE I illustrates that Material Interface Agent dealing with interface and input output related aspects; Material Security Agent dealing with security issues ; Material Monitoring Agent coping with registration and deregistration of agents should start with first iteration and continue to carry out for successive iterations as well.

Goal Oriented Agents such as Inventory Analyst, Inventory Controller and Store Receiver Agent are processed in the first iteration to make one functional release of MM MAS.

The remaining agents including Indent Receiver, Disposal Agent and Communicative Agents are treated in subsequent iterations assisting successive releases.

This paper discusses implication of proposed requirements model on the Goal Oriented agents associated with Inventory Control only and similarly same process can be extended for other agents attributed to other activities.

Implication of proposed Requirement Methodology on inventory control activity of MM MAS involves the following steps:

3.1 Requirements Elicitation

(i) Acquiring USCs:

To deal with requirements of Inventory control activity, various users enter their requirements in the above mentioned user story template (a):

1) *As an* Inventory Analyst, *I Want* to classify items as 'high value', 'medium value' and 'low value' *So That* I could control the 'high value' and 'medium value' items strictly.
2) *As an* Inventory Controller, *I want* to get report for items which are at reorder level *So That* order can be placed.
3) *As an* Inventory Controller, *I want* to get report on economic quantity *So That* I know how much quantity should be ordered.
4) *As an* Inventory Analyst, *I want* to know the list of non moving items *So That* these can be disposed off.
5) *As a* Stock handler, *I want* to retrieve the stock *So That* I know the level of stock at a given point of time.
6) *As a* Inventory handler, *I want* to manage the stock *So That* I can control the inventory.
7) *As a* Stock handler, *I want* to receive and issue the stock *So That* I can maintain stock.

All these user stories are written on physical cards (and later on can be stored in some electronic cards) or directly USCs mentioned in *Fig. 1* can be used to facilitate the users to feed stories and related parameters (like date, user story title, story points, execution time etc.).

The above requirements associated with various User stories can be finally placed in respective USCs as shown below:

USC_1: Classify among high, medium and low value

USC_2: Get report for items which are at reorder level

USC_3: Get report on economic quantity

USC_4: Obtain list of non moving items

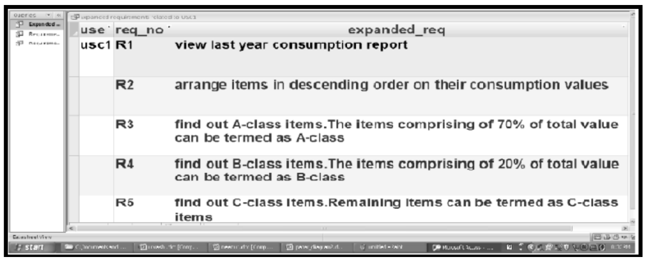
USC_5: Retrieve the stock

USC_6: Manage stock

USC_7: Receive and issue stock

(ii) Expansion of Requirements

User Story piled up with USC_1 after having a dialogue with users is intensified into a number of related requirements according to their semantic meanings. The USC_1 is expanded into the requirements R_1, R_2, R_3, R_4, R_5 given as below:



use	req_no	expanded_req
usc1	R1	view last year consumption report
	R2	arrange items in descending order on their consumption values
	R3	find out A-class items.The items comprising of 70% of total value can be termed as A-class
	R4	find out B-class items.The items comprising of 20% of total value can be termed as B-class
	R5	find out C-class items.Remaining items can be termed as C-class items

Fig 7: Expanded requirements associated with USC_1

In a similar manner, USC_2, USC_3, USC_4, USC_5, USC_6, and USC_7 can be expanded as shown below:

USC_2:
R_6:- Extract maximum quantity consumed per day during last one year.
R_7:- Find out ROL (Reorder Level) as:
ROL= (maximum quantity consumed per day) * max-lead-time
R_8:- Get report for items which are at reorder level.

USC_3:
R_1:-View last year consumption report (Redundant requirement)
R_9:- Get last year consumption value (S) of the items.
R_{10}:- Compute ordering cost (O) and carrying cost (C)
R_{11}:- Get report on economic quantity computed by formula:

$$EOQ=\sqrt{(2*S*O/C)} \qquad (3)$$

USC_4:
R_{12}:- Display the last issue dates item wise.
R_{13}:- Implicate a check constraint whether last issue date is the date of previous year.
R_{14}:- Display list of non moving items.

USC_5:
R_{15}:- Retrieve stock entries.

USC_6:
R_{15}:- Retrieve stock entries (Redundant requirement)
R_{16}:- Update stock
R_{17}:- Add stock
R_{18}:- Delete stock

USC_7:
R_{19}:- Receive stock
R_{20}:- Issue Stock
R_{21}:- Create pallets
R_{22}:- Process serial numbers
R_{23}:- Receive release order from purchase agent

3.2 Requirements Definition

In this step, developer carries out mapping from USCs to requirement R to have an accurate impression of unique requirements pertaining to different USCs. Additionally, this mapping facilitates the developer to revive his goals in subsequent phases as well as assists developer in documentation.

As a result of mapping USCs$\rightarrow$R developer comes across the piece of information that R_1 is associated with USC_1 as well as to USC_3. Likewise R_{15} is associated with USC_5 and USC_6. This consolidated list of USCs and requirements assists the developer to remove redundant requirements.

The developer extracts requirements from the USCs and assigns to various agent roles. All through this course of action, developer procures the requirements one by one from the list and allocates the functionality in terms of goals, tasks to numerous agent roles. Relation Ar leads to identification of following agent roles w.r.t. AC_1, AC_2, AC_3, AC_4 and their associated requirements:

Agent role:	*Requirements*
Inventory Controller (AC_1)	R_6, R_7, R_8, R_1, R_9, R_{10}, R_{11}, R_{15}, R_{16}, R_{17}, R_{18}
Inventory Analyst (AC_2)	R_1, R_2, R_3, R_4, R_5, R_{12}, R_{13}, R_{14}
Store Receiver Agent (AC_3)	R_{15}, R_{17}, R_{19}, R_{20}, R_{21}, R_{22}
Store Agent (AC_4)	R_{23}

Relation U_r and A_r facilitate developer to achieve mapping U_a to have a view of consolidated association of USCs and ACs in the following manner:

ACs	*Associated USCs*
AC_1:	USC_1, USC_2, USC_3, USC_6
AC_2:	USC_1, USC_4
AC_3:	USC_6, USC_7
AC_4:	USC_7

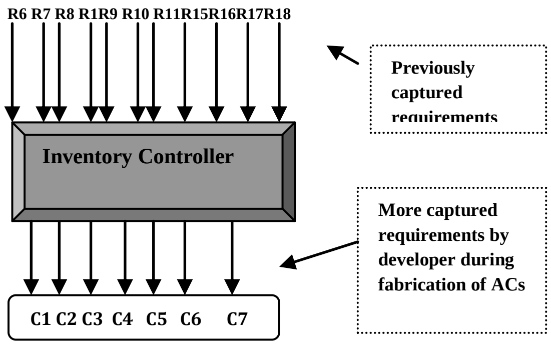

R6 R7 R8 R1R9 R10 R11R15R16R17R18
Previously captured requirements
Inventory Controller
More captured requirements by developer during fabrication of ACs
C1 C2 C3 C4 C5 C6 C7

Fig 8: Captured requirements w.r.t. Inventory Controller

Above association helps in defining requirements by facilitating a pertinent and unambiguous list of captured USCs so as to lead in fabrication of ACs. In above association USC_5 is not mapped to any of ACs. This means that this USC_5 is having only redundant requirements such as R_{15} which already has been covered up by some other user story card (USC_6). Thus this mapping helps to eradicate USCs having redundant requirements.

During this entire course of action, developer captures additional requirements from his experience. For inventory control system, following Goal Oriented agent role and additionally captured requirements are worked out:

Developer associates the requirements R_6, R_7, R_8, R_1, R_9, R_{10}, R_{11}, R_{15}, R_{16}, R_{17}, R_{18} taken from USC_2, USC_3, USC_5, USC_6 leading to goals G_1, G_2,G_3 w.r.t. agent role Inventory Controller (linked with AC_1) as:

G_1: Get report for reorder level
G_2: Get report on economic quantity
G_3: Manage stock

Now additional requirements C_1, C_2, C_3, C_4, C_5, C_6, C_7 are captured which otherwise are overlooked in initial stage of requirements elicitation.

C_1:- In equation (3), when C happens to be zero, developer decides to set the action as “Input value of C other than zero”
C_2:- Reorder level should always be greater than minimum level.
C_3:- Run a series of inventory reports daily, weekly or monthly.
C_4:- Retrieve stock, date wise in sorted order.
C_5:- Receive ABC analysis report from inventory analyst.
C_6:- Perpetual updation of equipment records.
C_7:- Maintain location files.

In a similar manner, developer extends the process of ACs for other Agent Roles such as Inventory Analyst (linked with AC_2) and Store Receiver Agent (linked with AC_3) selected for the recent iteration so as to release one functional version. Additional requirement R_{22} associated with Store Agent is kept with red mark in agent catalogue for later processing.

3.3 Requirements Validation

As prescribed in proposed methodology, a reverse mapping is processed to achieve completeness and consistency in the goals of ACs and USCs in following manner:

For AC_1, number of tasks corresponding to associated requirements are designated as: T_1: R_1, T_2: R_6, T_3: R_7, T_4:R_8, T_5: R_9, T_6:R_{10}, T_7:R_{11}, T_8:R_{15}, T_9:R_{16}, T_{10}: R_{17}, T_{11}:R_{18}, T_{12}:C_1, T_{13}:C_2, T_{14}:C_3, T_{15}:C_4, T_{16}:C_5, T_{17}:C_6, T_{18}:C_7

Check condition (1) for AC_1, as condition (1) is turned out to be false, then for condition (1.1):
For AC_1:
No. of mapped tasks=11
No. of non-mapped tasks=7
For every task of AC_1, there is not corresponding goal. At this point developer from his own understanding assigns goals to non mapped tasks as below:

Non mapped Tasks	*Assigned Goals*
T_{12}: T_{13}:	Exceptional and Functional Constraints
T_{14}: T_{15}:	Aspects related to Inventory Report
T_{16}: T_{17}: T18:	Manage Inventory Data

Now as equation (1.1) comes out to be true, developer checks for condition (1.2) by comparing each and every goal of AC_1 to the initially elicited goals in USCs:
For AC_1:

$$\text{for } G_1 \in AC_1 \;\; \exists USC_2$$
$$\text{for } G_2 \in AC_1 \;\; \exists USC_3$$
$$\text{for } G_3 \in AC_1 \;\; \exists USC_6$$
$$\text{for } G_4 \in AC_1 \;\; \nexists U_h$$
$$\text{for } G_5 \in AC_1 \;\; \nexists U_h$$
$$\text{for } G_6 \in AC_1 \;\; \nexists U_h$$
$$\therefore \;\text{for } AC_1 \text{ no. of non mapped goals} = 3$$
$$\Rightarrow \quad GE_1 = 0$$
$$\Rightarrow \quad VF = 0 \qquad (\because \; VF = GE_1 * GE_2 * GE_3)$$

Subsequently developer extends the same process for AC_2 and AC_3 linked with Inventory Analyst and Store Receiver Agent respectively. And if for atleast one AC, value of GE comes out as 0 subsequently leading to the value of VF(Validation Factor) as 0, developer is required to conduct a workshop with users so that USCs can be updated according to these new captured requirements with the consent of users. This process is iteratively executed till all users and developers are satisfied and value of VF comes out as 1. This way, a complete, consistent, unambiguous and unique list of requirements results as a baseline for the system.

4. CONCLUSIONS

This work extends Agent-Oriented approach to requirements engineering using USCs and ACs. Methodology presented in this work captures user-requirements in the form of USCs that are mapped to ACs. Representing requirements in the form of USCs is a user oriented view of requirements engineering, while ACs is a developer oriented view of requirements definition. By incorporating ACs, this methodology assists the developer to obtain complete, unambiguous, as well as consistent requirements and also ensures that users' expectations are truly reflected in the final requirements document. The system that is developed on these requirements is more close to users' expectation and will eventually result in their satisfaction. Further study is required to prove that the system built using the proposed methodology results in higher users' satisfaction.

5. FUTURE DIRECTIONS

This works applies reverse mapping from ACs to USCs for validating the requirements. However its complement activity namely verification would be considered for future study.

6. REFERENCES

[1] Singh,Y. Gosain,A. Kumar,M. Evaluation of Agent Oriented Requirements Engineering Frameworks, IEEEVolume:2, pp. 33-38

[2] Regev, G.; Wegmann, A., Where do goals come from: the underlying principles of goal-oriented requirements engineering, 2005. Proceedings. IEEE,pp.353-362

[3] A M Sen, S K Jain, An Agile Technique for Agent Based Goal Refinement to Elicit Soft Goals in Goal Oriented Requirements Engineering, ADCOM 2007.International Conference IEEE, pp.41-47

[4] Luiz Marcio Cysneiros, Requirements Engineering for Large-Scale Multi-Agent Systems, LNC, Vol. 2603, 2003, PP.77-148

[5] Paolo Donzelli, REF: A Practical Agent-Based Requirement Engineering Framework, Springer, 2003, pp.217-228

[6] Ruben Fuentes, Requirement Elicitation for Agent Based se Cases based Requirements validation With Scenarios, 2005, IEEE, pp. 465- 466

[7] Michael j Rees, A Feasible User Story Tool for Agile Software Development, IEEE ,pp 22, 2002

[8] Mike Cohn, Agile Estimating and Planning, Pearson Education, 2006, ISBN 978-81-317-0548-3

[9] Connolly, D., Keenan, Tag Oriented Agile Requirements Identification, ECBS 2008. IEEE, pp:497 – 498

[10] PABRE: Pattern-Based Requirements Elicitation, IEEE

[11] Eliciting and Specifying Requirements with Use Cases for Embedded Systems, 2002, IEEE

[12] Computer Assisted and Customer-Oriented Requirements Elicitation,2005, IEEE

[13] A New Approach for Software Requirements Elicitation, 2005, IEEE

[14] The Value Gap Model: Value-Based Requirements Eliciation,2007 IEEE

[15] A. Duran, B.Bernardez "A Requirements Elicitation Approach Based in Templates and Patterns". In Proceedings 2nd Workshop on Requirements Engineering (WER'99), 1999

[16] Gaur, Vibha, AnujaSoni and PunamBedi. 2010. An agent-oriented approach to requirements engineering. In proceedings 2010, IEEE, 449-454.