The Design of the Multi-Agent Classification System (MACS)

Mohamed R. Mhereeg

Abstract**—**The paper discusses the design of a .NET Windows Service based agent system called MACS (Multi-Agent Classification System). MACS is a system aims to accurately classify spreadsheet developers competency over a network. It is designed to automatically and autonomously monitor spreadsheet users and gather their development activities based on the utilization of the software multi-agent technology (MAS). This is accomplished in such a way that makes management capable to efficiently allow for precise tailor training activities for future spreadsheet development. The monitoring agents of MACS are intended to be distributed over the WWW in order to satisfy the monitoring and classification of the multiple developer aspect. The Prometheus methodology is used for the design of the agents of MACS. Prometheus has been used to undertake this phase of the system design because it is developed specifically for specifying and designing agent-oriented systems. Additionally, Prometheus specifies also the communication needed between the agents in order to coordinate to achieve their delegated tasks.

Keywords**—**Classification, Design, MACS, MAS, Prometheus.

I. Software Development Methodology for Agent-Based Systems

AS agents are gaining acceptance as a technology and are being used, there is a growing need for practical methods for developing agent applications. However, one of the most fundamental obstacles to large-scale take-up of agent technology is the lack of mature software development methodologies for agent-based systems [8]. Numerous agent oriented methodologies have been proposed in the literature such as Prometheus [5], [6], GAIA [13], TROPOS [10], [2] and MaSE [11], [12], just to name a few. Additionally, there are other software development methodologies such as Agile, RUP, and Shlaer-Mellor that can be considered as alternatives for the design of the agent system. The section below briefly explains why Prometheus has been chosen for the design of the agent system.

II. The Prometheus Methodology

Prometheus is intended to be a *practical* methodology. As such, it aims to be complete: providing everything that is needed (start-to-end process) to specify and design agent systems. Other distinguishing features of the Prometheus methodology are [7]:

- Prometheus is *detailed* – it provides detailed guidance on *how* to perform the various steps that form the process of Prometheus.
- Prometheus supports (though is not limited to) the design of agents that are based on goals and plans.
- Prometheus covers a range of activities from requirements specification through to detailed design.
- Prometheus allows the creation of the communication between the agents when a message is sent from one agent to another.
- The methodology is designed to facilitate tool support, and tool support exists in the form of the *Prometheus Design Tool* (PDT).

Prometheus, through its three phases (Fig. 1), system specification, architectural design, and detailed design, defines an in-depth process for specifying and designing agent-oriented systems. The process defines a range of artefacts some of which are used as permanent, and others that are used as 'stepping stones' for other artefacts. Artefacts include goals, capabilities, events, plans and data structures, actions and percepts.

The system specification phase focuses on the identification of system goals, developing use case scenarios that demonstrate the operation of the system to be developed, determining the basic functionalities of the system and the specification of the interface between the system and its working environment by identifying the result actions.

The architectural design phase builds on the artefacts (deliverables) from the system specification phase to determine the composition of the agents the system will contain and how they will interact. This stage is also used to capture the system's overall structure and its dynamic behaviour.

The detailed design is used to establish the internal design of each agent within the system and is not dependent on any one particular development platform/environment.

Although Agile, RUP, and Shlaer-Mellor are general purpose software development methodologies, these methods could be used as alternative solutions for the design of the agent system. In contrast, Prometheus is developed specifically for specifying and designing agent-oriented systems, therefore it has been chosen to undertake this phase of the project. Additionally, Prometheus and its accompanying tool (PDT) beyond specifying and designing the agents required to construct the agency specify also the communication needed between these agents in order to coordinate to achieve their delegated tasks. Furthermore, Prometheus has been compared to other existing software development methodologies for agent-based systems [3], [4].

M. R. Mhereeg was with Glamorgan University, Pontypridd, CF37 1DL, UK. He is now with the Department of Computer Science, Tripoli University, Tripoli, Libya (e-mail: mmhereeg@msn.com).

The features of Prometheus distinguish it from these methodologies, but none of them have all of the Prometheus features described below [7].

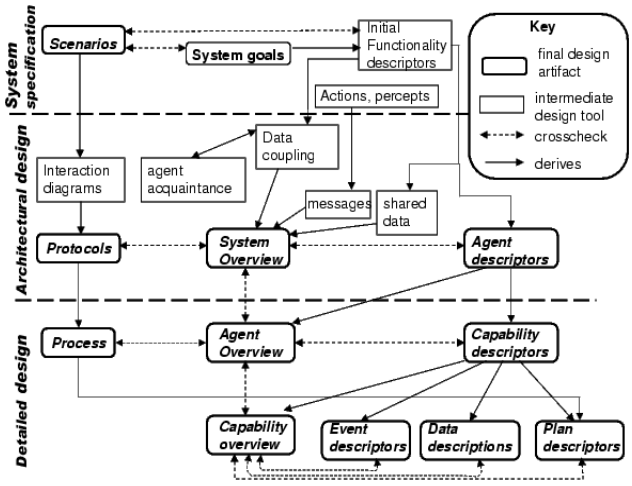

Fig. 1 The phases of Prometheus methodology

Thus, Prometheus methodology and its accompanying Prometheus Design Tool (PDT) have been chosen for specifying and designing the agents needed to construct the Multi-Agent System (MACS). However, above is a demonstration of how Prometheus through its three phases (Fig. 1), system specification, architectural design and detailed design can be used to achieve this goal.

A. System Specification

This section discusses the artifacts and processes required in the system specification phase, the initial phase of the Prometheus methodology.

1. Goal Specification

a. Brief Statement of the Problem Scenario

The system will automatically and continuously monitor end-users developing application in the spreadsheet domain over a network and gather their development activities. The data gathered from multiple resources will be collected in an Oracle Server Database for further processing. The processing will lead to the classification of the users based on well-defined classification criteria explained in Table I. The classification is dependent of their knowledge and experience of both excel and the excel macro language Visual Basic for Application (VBA) [1]. The categories are Excel User, Excel Power User, VBA Developer, Excel Developer, and Professional Excel Developer. A typical excel user may start off as a simple consumer of data and progress through simple charts, formulas, pivot tables, VBA development, and progress further to automate other applications, use Windows API calls and XML for sharing data over the Internet. Table I adapted from [1] summarizes the categories of the users. It explains the categories together with a description and the level of knowledge (Usage) required for every category.

The MultiAgent Classification System (MACS) will consist of a number of software agents. These agents will monitor the usage of software applications on end-users' machines with any new usage patterns being recorded. The monitoring agents will be installed manually on the users' computers and when Excel spreadsheets are executed, the agents will listen and gather the contents of these applications.

The monitoring agent will also detect when a macro/Visual Basic Application (VBA) code has been recorded and report when features are being used, such features in the program environment include the number and type of: selections, sequences, iterations, procedures, functions, inclusion of ActiveX controls and references to COM object libraries. The agents of MACS will be distributed on the user's computers over the network and run autonomously in the backgrounds collecting the data as long as these computers are being used by the end users.

TABLE I
EXCEL DEVELOPER CATEGORIES

CATEGORY	DESCRIPTION
Excel User	- Store lists, Simple repetitive calculations - Some worksheet functions - Pivot tables - Charts
Excel Power User	- Wide understanding of excel Functionality - Create complex spreadsheets for own use and helps develop and debug colleague's spreadsheets. - Occasional use of VBA code from macro recorder or the Internet.
VBA Developer	- Extensive use of VBA - Typically they are Power Users who started to learn VBA or Visual Basic developers who switched to VBA development - Often lack sufficient knowledge of Excel to make the best use of its features.
Excel Developer	- Constructs efficient and maintainable applications by making the best use of excels' built-in functionality, augmented by VBA when appropriate. - Confident at developing excel based applications for both their colleagues and as part of a development team. - Constrained by their reluctance to use other programming languages and applications to augment their excel solutions.
Professional Excel Developer	- Design and develops excel based applications and utilities that are robust, fast, easy to use, maintainable and secure. - Excel forms the core use of their applications, but they include any other applications and languages that are appropriate For Example: - They might use third party ActiveX controls - Automate other applications. - Use Windows API calls. - Use ADO to connect to external Databases. - Use C/C++ for fast custom worksheet functions (DLL/XLL add-ins in XLM macro sheets). - Use VB6 or VB.net for creating object -modules and securing their code. - Use XML for sharing data over the Internet.

b. Identifying Initial Goals

The system is intended to be developed as a multi-agent based that can automatically monitor the software usage of Excel users' development activities on an individual basis and categorize the usage obtained into different categories using well-defined classification criteria from the literature (Table I). The activity together with the categorization can then be used to recommend training and mentoring in order to make the user more productive with respect to the end-user application development. The system must be able to identify

any new application development patterns have occurred and distinguish them from pre-existing patterns; this includes the detection of any Macro/Visual Basic Application (VBA) codes generated by the user. The system must be reliable and gather data from both spreadsheets under development and developed applications and produce the data mining based categorization results either on demand or at regular intervals, i.e. monthly or three monthly. This step requires:

- Identify System Goals

- Monitor software usage of end-users
- Identify application development
- Identify new usage patterns
- Create users Data Mining mapping
- Identify end-users categories

- Goal Refinement

Goal refinement leads to the question 'how might each goal be achieved'? Answering the question normally requires identifying the sub-goals of the goal under consideration. As the initial goals are refined, similar sub-goals may appear under different parent goals. Grouping these sub-goals provides the basis of what is called 'functionalities' or 'roles' – chunks of behavior of the system. Below shows the expanded list of the initial goals and associated sub-goals, and the initial arrangement of these into groupings.

- Monitor software usage of end-users
  - Identify users
  - Identify packages used
  - When packages used
  - Obtain usage level
- Identify application development
  - Track creation of programme code
    - Track automatic code generation
    - Track generation of hand coding
- Identify new usage patterns
  - Obtain new object usage
  - Obtain code structures
    - Determine new code structures
    - Determine existing code structures
- Create users data mining analysis
  - Get users development efforts
  - Calculate development plane
- Identify users categories
  - Get skills profile of end users
  - Produce categorization results

After refining of goals takes place; these goals have then been rearranged, moving similar goals together, and adding some extra goals to some particular groupings that are lacking some aspect. As this task is achieved, the original set of goals and sub-goals form a network of connected goals. Goals are represented by ovals and arrows join goals to sub-goals. Fig. 2 shows the goal overview diagram for the MultiAgent Classification System.

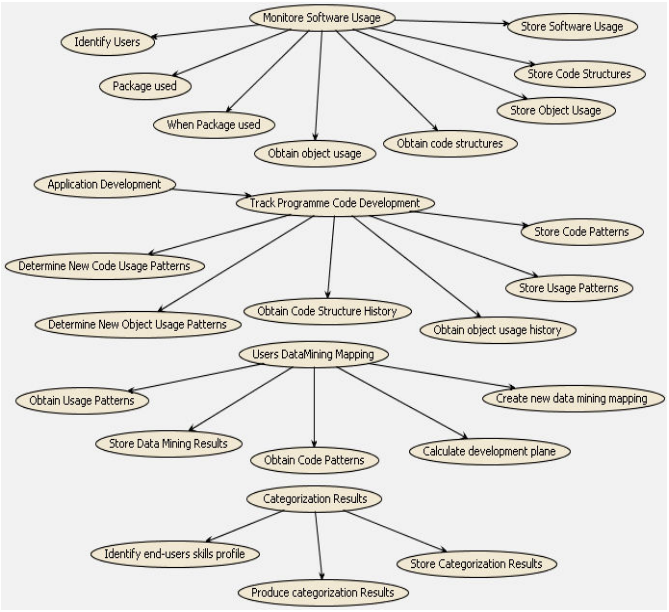

Fig. 2 Goal Overview Diagram

2. Roles

Roles can be described as a block of behavior, which includes a grouping of goals, as well as percepts, actions and data related to the behavior [9]. The input is represented as a percept and the output as an action. However, grouping of related roles can eventually lead to determining the agent types and their responsibilities. Roles are described by rectangles, goals with ovals, and actions with a rectangle extended with a triangle pointed to the right.

At this stage goals are grouped into cohesive units and assigned to roles which are intended as relatively small and easily specified chunks of agent functionality. The percepts and actions are then also assigned to the roles appropriately to allow the roles to achieve their goals. This is done using the 'System Roles Diagram'.

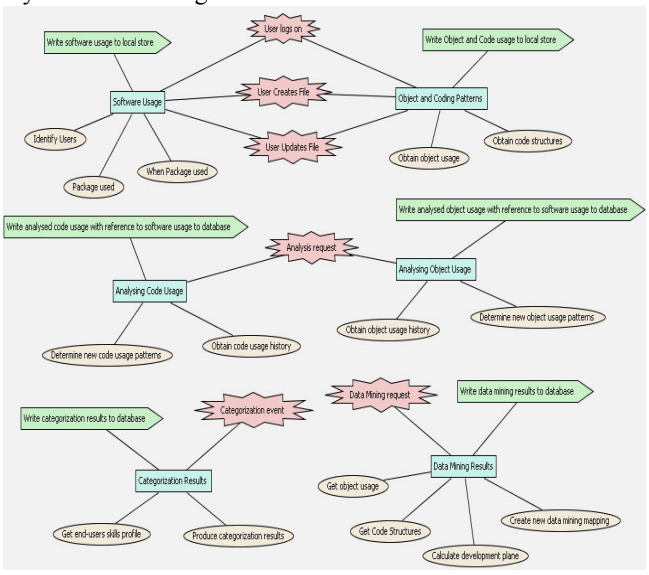

Fig. 3 System Roles Diagram

Fig. 3 above shows the "*Software Usage Role"* is responsible for the goals to *Identify Users, Packages Used* and*, When Packages Used* (i.e. File Properties) of any files used by end users. To achieve this goal the role needs the inputs *User logs on, User Creates File*, or *User Updates File* and should perform the action *Write Software Usage to a local store.* Once roles have been identified, role descriptors can be developed. The below shows the role descriptors have been developed on the basis of the goals grouped under each role.

The percepts *User logs on, User creates file and User updates file* all trigger two roles: "*Software Usage Role" and "Object & Coding Patterns Role"*. They carry out the actions: *Write software usage to a local store and Write Object and Coding patterns to local store* respectively, and depending on the percept the appropriate goals will be satisfied.

Software Usage Role Descriptor:

Name Software Usage
Description The role monitors the software being used on a daily basis by end users
Percepts User logs on, User creates file, User updates file
Actions Write software usage to a local store
Data used Software used
Data produced Software used local store
Goals Identify users, Package used, When packages used

Object and Coding Patterns Role Descriptor**:**

Name Object and Coding Patterns
Description The role monitors the Object and Code patterns of a software being used on a daily basis by end users
Percepts User logs on, User creates file, User updates file
Actions Write Object and Coding patterns to a local store
Data used Object and coding patterns used
Data produced Object and coding patterns used local store
Goals Obtain object usage, obtain code structures

The percept *Analysis Request* is performed in an ad-hoc manner to trigger the roles "*Analysing Object Usage" and "Analysing Code Usage patterns"*. They carry out associated actions of *Writing analysed object usage with reference to software usage to a database* and *Writing analysed code usage with reference to software usage to a database* respectively to meet the roles' assigned goals.

Analysing Object Usage Role Descriptor:

Name Analysing Object Usage
Description For each user build up a history of the object usage and determine new object usage patterns
Percepts Analysis request (ad-hoc request)
Actions Write analysed object usage with reference to software usage to a *database*
Data used Object usage patterns used
Data produced Object usage patterns used database
Goals Obtain object usage history, determine new object usage patterns

Analysing Code Usage Role Descriptor:

Name Analysing Code Usage
Description For each user build up a history of the code usage and determine new code usage patterns
Percepts Analysis request (ad-hoc request)
Actions Write analysed code usage with reference to software usage to a *database*
Data used Code usage patterns used
Data produced Code usage patterns used database
Goals Obtain code usage history, determine new code usage patterns

The ad-hoc *Data Mining Request* percept triggers the role "*Data Mining Analysis"*. The action *Write data mining results to database* is to be taken to satisfy the goals assigned to this role.

Data Mining Analysis Role Descriptor:

Name Data Mining Results
Description Get object and code usage patterns of users and calculate development plane
Percepts Data Mining request (Ad-hoc request)
Actions Write data mining results to a *database*
Data used Analysed object and code usage patterns
Data produced Data mining results database
Goals Get object usage, Get code usage, calculate development plane, create new data mining mapping

The percept *Categorization Event* triggers the role "*Categorization Results"* that is performed after the "*Data Mining Analysis Role"* has been achieved. The end users skills profile is retrieved and examined. Resulting in, a categorization results are to be produced that shows the users' categories have been developed over a period of time.

Categorization Results Role Descriptor:

Name Categorization Results
Description Obtain skills profile of users for categorization purposes
Percepts Categorization event (Ad-hoc request)
Actions Write categorization results to a *database*
Data used End-users skills profile database
Data produced Categorization results database
Goals Get end-users skills profile, produce categorization results

3. Interface Description

Agents are often situated in an environment that is dynamic and change rapidly. Thus, agents need to respond to these changes while trying to achieve a goal. Interface description deals with how agents interact with and affect the changing of the environment in which they are situated.

a. Percepts and Actions

Percepts can be defined as the expected incoming information that will be available while the agent is running, the agents in this case are required to react to this change in the environment and action is to be taken to fulfill the task under consideration. Two types of incoming information have been considered. The first one happens via some form of event (user logs on a computer), in the second the agent is required to seek the data via monitoring the environment and listening to any files accessed by the users.

The list below illustrates the percepts and actions identified for the MACS system.

Percepts:
- User logs on
- User creates file
- User updates file

Actions:
- Store software usage
- Store object & code usage patterns
- Store data mining results
- Store categorization results

b. Data

As roles are developed, it is also important to consider both the data is to be created or consumed. Data produced and used by MACS system are:

Software Used Database - contains data relating to the software applications being used by each end user, e.g. Author, Date Created, Date Last Saved, Last Saved By,.. etc. it also contains information on the object patterns and code structure present in end users code, e.g. selection statements, iterations, procedures, functions (Table II shows the software usage and object & code patterns). This database keeps track of the users' development activities. *Software Usage Analysis Database* - contains the total analyzed usage of the users' object and code patterns of developed applications. This usage will be used for mining the users' development activities.

Data Mining Results Database - contains records of end users skills profile. *Categorization Results Database* - contains information relating to end users categorization results.

TABLE II
SOFTWARE USAGE AND OBJECT & CODE PATTERNS

Number Of Functions	**Number Of Iterations**
Database	Do … Loop Until
Lookup & Reference	Do … Until Loop
Date & Time	Do… Loop While
Math & Triggers	Do …While Loop
Financial	For … Next
Information	For … Each
Logical	**Number of Selections**
Text	If … Then … End If
Cube functions	If … Then … Else … End If
Engineering functions	Select … Case
Add-ins and automaton	**File Properties**
Functions	
Calculate number of variables and constants	Author
Constants	Company
Variables	Version
Calculate number of Procedures	Date Last Saved
Function	Date Created
Sub	Comments
Applications & Languages	Document Name
Automated applications	Path Name
Windows API calls	Application
ADO	Last Saved By
C/C++, XML	Title
VB6 or VB.net	Subject

B. Architectural Design

In the System Specification phase, the system scenarios, goals, roles and role descriptors are developed. In the architectural design phase these artifacts will be used for developing the high-level design of the agent system. In Prometheus, all phases of the design interact with each other. Therefore, the architectural design aspects in this phase interact with each other as well as with the system specifications phase features.

The three aspects that are developed during architectural design are:
1. Deciding on the agent types used in the application.
2. Describing the interaction between agents.
3. Designing the overall system structure using the *System Overview Diagram*.

1. Deciding on the Agent Types

A major decision is to be made during the architectural design is deciding the agent types the system will contain. Agent types are formed by combining roles, taking into consideration the standard software engineering criteria of coupling and cohesion. The relationships between roles and data stores they interact with are also considered in determining the agent types.

a. Grouping Roles

After roles have been identified, there are many possible solutions in which these roles can be grouped into agents. A good proposed grouping is the one that offers lower degree of dependency or coupling and higher degree of cohesion. [9] Stated two reasons for deciding to group roles into a single agent:
1. The roles seem related – it 'makes sense' to group them. For example, the *Software Usage* and *Object & Coding Usage* roles are clearly related.
2. The roles require a lot of the same information. If grouped into a single agent. This can then be represented in internal agent data structure.

They also stated the below reasons for not grouping roles:
1. The roles are clearly unrelated.
2. The roles exist on different hardware platforms.
3. Different numbers of roles are required at run time.

The Data Coupling Diagram is the tool used to develop and assess the proposed groupings. These groupings are then evaluated and refined using the Agent Acquaintance Diagram.

1) Data Coupling Diagram

One technique that is used to assess coupling and help in finding groupings which are relatively loosely data coupled is the Data Coupling Diagram. A data coupling diagram consists of all roles formed in the previous phase linked to data that has been identified as necessary for performing the role (not only persistent data, but also data the roles require to fulfill their mission).

Rectangles are used to depict roles, and cylinders are to depict data. Direct links are then inserted between roles and data, an arrow from role linked to data indicates the data is produced or written by the role, whereas an arrow from data linked to role indicates that data is used or read by that role. A double-headed arrow indicates that the role both produces and uses the data.

Various design decisions can be made to group roles into agents. A robust rationale for grouping roles is to examine the data sharing, and in particular if roles write to the same database store, it is an indication for grouping them, thus these roles should be in the same agent. Fig. 4 shows the Data Coupling Diagram with two possible groupings.

The roles *Software Usage* and *Object & Coding Usage Patterns* are both used to populate the *Software Usage Local Data Store*, thus they make up the first agent namely the Monitoring Agent. The roles *Analysing Object Usage* and *Analysing Code Usage* Patterns read data from the *Software Usage Local Data Store,* analyse the data and, then populate the *Software Used Database* and *Software Usage Analysis Database* data stores. These roles make up the second agent named Database Updater Agent.

The *Data Mining Analysis* and *Categorization Results* Roles are not to be grouped. These roles will be performed manually and separately by the User Agent (i.e. Administrator of the system) in an Ad-hoc manner.

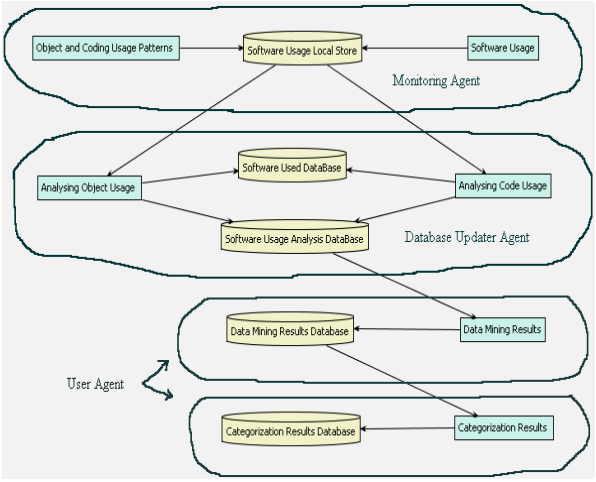

Fig. 4 Data Coupling Diagram

2) Agent-Role Grouping Diagram

In this diagram the roles are grouped into agents. The Agent-Role Grouping Diagram shows the group of roles that come under each agent. Fig. 5 illustrates that the roles of *Software Usage* and *Object & Coding Usage* as being part of the Monitoring Agent, whereas the roles of *Analysing Object Usage* and *Analysing Code Usage* as being part of the Database Updater Agent.

Fig. 5 Agent-Role Grouping Diagram

3) Agent Acquaintance Diagram

The Agent Acquaintance Diagram evaluates the groupings for coupling and places a link where there is a communication between the agents. These links are automatically created when a message (call) is sent from one agent to another. Fig. 6 shows that a message is sent from the Database Updater Agent to the Monitoring Agent requesting the collected data the local store contains.

One of the design aims is to produce a system that is as loosely coupled as possible. The figure below indicates that the design has a low level of coupling with one linkage between the agents and therefore preferable.

Fig. 6 Agent Acquaintance Diagram

2. Developing Agent Descriptors

Once the agents of the system have been decided, Agent descriptors are to be produced. This would provide higher level information contains answers to the following questions

- How many agents of each type will there be?
- What is the lifetime of the agent?
- Agent initialization – what needs to be done?
- Agent demise – what clean up needs to be done?
- What percepts will this agent react to?
- What actions will it take?
- What are the goals of the agent?
- What data does this agent uses or produces?

In addition, each agent should have an extra information includes the name of the agent, a natural language description of what this agent does within the system and a list of the roles that have constructed the agent. Tables III and IV show descriptors for the Monitoring Agent and the Database Updater Agent respectively.

TABLE III
THE MONITORING AGENT DESCRIPTOR

Name	Monitoring Agent
Description	Monitor software packages used by an end user
Cardinality minimum	One per networked computer
Cardinality maximum	One per networked computer
Lifetime	Instantiated when user logs on. Demise when user logs out, Can be started, paused, stopped and restarted manually by the user
Initialization	Listen for packages that are being launched, data collection
Demise	Close open communications
Percepts	User logs on, user creates file, user updates file
Actions	Write software usage to local store, write object & Coding patterns usage to local store
Uses data	None
Produces data	Software usage local store
Goals	Identify users, packages used, when packages used, obtain object usage, obtain code usage
Roles	Software usage, object & code usage patterns

TABLE IV
THE DATABASE UPDATER AGENT DESCRIPTOR

Name	Database Updater Agent
Description	Collects data gathered by the Monitoring Agents, filters the data, and then connects to Oracle server database for data upload.
Cardinality minimum	One per networked master computer
Cardinality maximum	One per networked master computer
Lifetime	Instantiate manually by the user agent. Demise when user logs out
Initialization	Obtains data gathered by the Monitoring Agents
Demise	Closes open database connections
Percepts	Analysis request
Actions	Write analysed data with reference to software usage to Oracle Database
Uses data	Software usage local store
Produces data	Software used database, software usage analysis database
Goals	Obtain object usage history, determine new object usage patterns, Obtain code usage history, determine new code usage patterns
Roles	Analysing object usage, analysing code usage

3. System Overview

Having identified the agents the system will include. The top level structure of the system can be produced using the System Overview Diagram (Fig. 7). This diagram brings together the agents, percepts, actions, and data stores and shows how they will interact to achieve the system goals. It shows which percepts and actions are associated with which agent. It shows also which data stores are used, which are produced and, which are both used and produced by which agent. Messages that can be sent between agents can also be shown in this diagram.

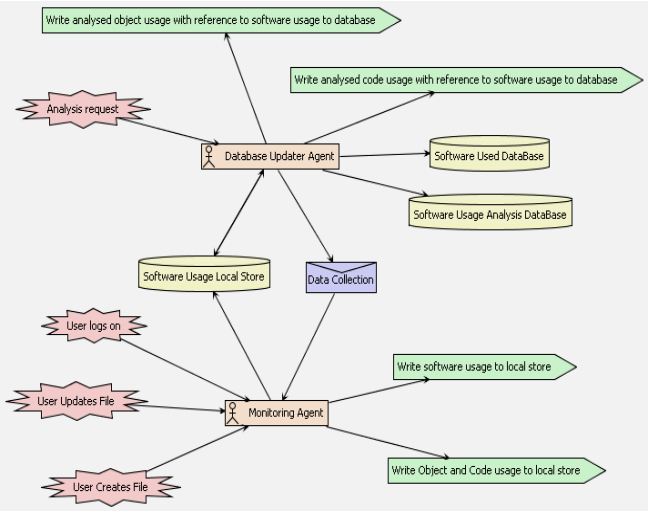

Fig. 7 System Overview Diagram

It is arguably the single most important artifact of the entire design process, although of course it cannot be really understood fully in isolation [9]. The System Overview Diagram assists to obtain a general understanding of how the system will function, including the interactions between agents.

C. Detailed Design

The detailed design phase deals with the internal behavior of each agent individually. It takes the artifacts have been developed in the architectural design phase and for every agent develops the capabilities necessary to achieve its goals. The roles of the agents that have been defined in the specification phase can be used to suggest an initial set of capabilities, which can then be refined as desired. The detailed design describes agents in terms of capabilities and the interactions between them. These capabilities are of necessity to the implementation platform more than the preceding steps. The detailed design phase consists of two Agent Overview Diagrams, one for each agent as shown below.

1. Agent Overview Diagram

The Agent Overview Diagram provides a top level view of the internal structure and behavior of the agents of the system. In this stage, an Agent Overview Diagram is produced for every single agent appeared in the System Overview Diagram. The Agent Overview Diagram is very similar to the System Overview Diagram, but instead of showing the interactions between agents within the system, it shows the interactions between the capabilities within an agent. This diagram brings together the capabilities, and all the percepts, actions and data stores linked to an agent within the system. The Agent Overview Diagram also includes the messages required between capabilities in order to realize the behavior of an agent. Figs. 8 and 9 illustrate graphically the detailed design of the Monitoring Agent and the Database Updater Agent respectively. By looking at these diagrams, it is possible to gain a high level view of how the capabilities of the system will interact to achieve the overall tasks of the agents.

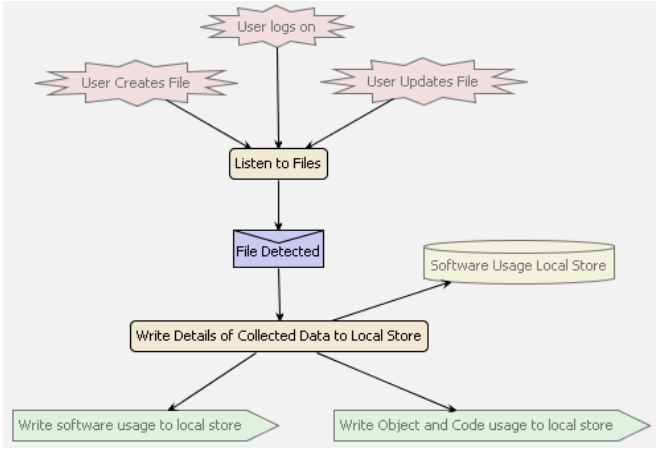

Fig. 8 Agent Overview Diagram for the Monitoring Agent

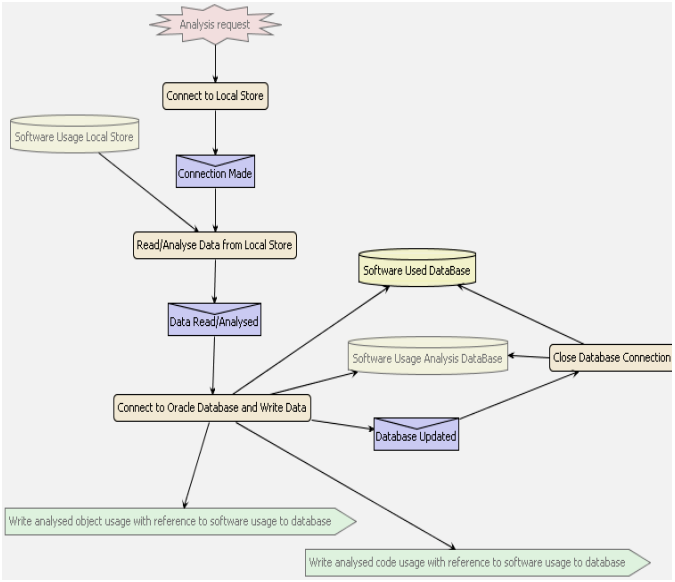

Fig. 9 Agent Overview Diagram for the Database Updater Agent

III. Conclusions

The Prometheus agent oriented methodology and its accompanying Design Tool (PDT) have been utilized effectively for the design of both the Monitoring and Database Updater agents of MACS. The PDT provided an instrument for cross checking the artifacts to ensure a consistent agent specification. Thus, the MA agents were designed with the capability to automatically and autonomously monitor spreadsheet development activities by content from different resources over a network, while the DUA agents were designed with the capability to retrieve and filter the data collected by the MA agents. As such Prometheus has proved to be practical and complete providing everything needed to specify and design the agents. Using Prometheus allowed the coverage of a range of activities from requirements specification through to detailed design. Additionally, Prometheus and it is tool (PDT) specified also the communication needed between the agents when a message is sent/received from one agent to another in order to coordinate to achieve their delegated tasks.

References

[1]. S. Bullen, R. Bovey, & J. Green, (2009), *Professional Excel Development: The Definitive Guide to Developing Applications Using Microsoft Excel and VBA,* Upper Saddle River, Addison-Wesley.

[2]. G. Fausto, M. john, and A. Perini, (2002), The Tropos software development methodology: Processes, models and diagrams. *In Third International Workshop on Agent-Oriented Software Engineering.*

[3]. H. D. Khanh, and W. Michael, (2003), Comparing agent-oriented methodologies. IN PAOLO, G. & MICHAEL, W. (Eds.) *Proceedings of the Fifth International Bi-Conference Workshop on Agent-Oriented Information Systems.* Melbourne, Australia.

[4]. H. D. Khanh, (2003), Evaluating agent-oriented software engineering methodologies, Master's thesis, *School of Computer Science and Information Technology.* Melbourne, Australia, RMIT University, (supervisors: Michael Winikoff and Lin Padgham).

[5]. P. Lin, and W. Michael, (2002), Prometheus: A methodology for developing intelligent agents. *Third International Workshop on Agent-Oriented Software Engineering.*

[6]. P. Lin, and W. Michael, (2002), Prometheus: A pragmatic methodology for engineering intelligent agents. *In Proceedings of the OOPSLA 2002 Workshop on Agent-Oriented Methodologies.*

[7]. P. Lin, and W. Michael, (2004), The Prometheus Methodology.

[8]. L. Michael, M. Peter, and P. Chris, (2003), Agent technology: Enabling next generation computing: A roadmap for agent-based computing. AgentLink report, available from www.agentlink.org/roadmap, ISBN 0854 327886.

[9]. L. Padgham, and M. Winikoff, (2004), *Developing Intelligent Agent Systems: A Practical Guide,* Chichester, Great Britain, Wiley.

[10]. B. Paolo, G. Paolo, G. Fausto, J. Mylopoulos, and A. Perini, (2002), Tropos: An agent-oriented software development methodology, Technical Report DIT-02-0015. University of Trento, Department of Information and Communication Technology.

[11]. A. Scott, and Deloach (2001), Analysis and design using MaSE and agent Tool. *In Proceedings of the 12th Midwest Articial Intelligence and Cognitive Science Conference (MAICS 2001).*

[12]. D. Scott, W. Mark, and S. Clint, (2001), MultiAgent systems engineering, *International Journal of Software Engineering and Knowledge Engineering,* 11(3)**,** 231-258.

[13]. M. Wooldridge, N. R. Jennings, and D. Kinny, (2000), The Gaia methodology for agent-oriented analysis and design. *Autonomous Agents and Multi-Agent Systems,* 3(3).