THE PROMETHEUS; A SPECIAL PURPOSE METHODOLOGY

BY

[1] Omankwu, Obinnaya Chinecherem;
Department of Computer Science,
Michael Okpara University of Agriculture,
Umudike Umuahia, Abia State, Nigeria.
saintbeloved@yahoo.com

[2] Anigbogu, S.O.
Department of Computer Science,
Nnamdi Azikiwe University,
Awka Anambra State,
Nigeria.
dranigbogu@yahoo.com

[3] Nwagu, Kenneth Chikezie
Department of Computer Science,
Nnamdi Azikiwe University,
Awka Anambra State, Nigeria,
Nwaguchikeziekenneth@hotmail.com

[4] Anigbogu, G.N.
Department of Computer Science,
Nwafor Orizu College of Education,
Nsugbe-Onitsha. Anambra State, Nigeria.
anigbogugloria@yahoo.com

Abstract

In this paper we presented the Prometheus methodology for building agent-based software systems. Our objectives in presenting Prometheus as a special purpose methodology were to have a process with associated deliverables which could be used by industry practitioners and undergraduate students without a previous background in agents' studies. Prometheus as an agent oriented methodology was developed based on the principles of Knowledge based engineering methodology. Prometheus comprised of three phases: system specification, architectural design, and detailed design. This special purpose methodology was used by industrial

practitioners, taught at workshops at a number of conferences, and has also been taught to undergraduate and postgraduate students, as well as used in student projects.

Keywords: Agents, Software Engineering, Methodologies.

1.0 INTRODUCTION

Prometheus is a methodology for developing agent-oriented software systems. Our goal in developing Prometheus was to have a process with associated deliverables which could be used by industry practitioners and undergraduate students to develop intelligent agents systems, without necessarily a previous background in agent's studies. Prometheus as a special purpose methodology covers only three phases of software development stages as applied to agent systems development.

The Prometheus methodology includes three phases:

The system specification phase: This focuses on (i) identifying the system's interface, which consists of percepts (information from the environment), and actions; and (ii) deter-mining the system's goals, functionalities, and use case scenarios, along with any important shared data. The outputs from this phase are set of functionality descriptions, percept and action descriptions, system goals, and use case scenarios (Bresciani et al., 2002).

The architectural design phase: This uses the outputs from the previous phase to determine which agents the system will contain, how they will inter-act, and what significant events occur in the environment. The outputs of this phase are system overview diagram, agent descriptions, agent interaction protocols and a list of significant events and messages between agents (Winikoff et al., 2001).

The three aspects that are developed during architectural design are:

1 Deciding on the agent types used in the application. Agent types are formed by grouping a number of functionalities together. Diagrams which we use to assist in the analysis are data coupling diagrams and agent acquaintance diagrams (Winikoff et al., 2001).

2. Designing the overall system structure (with a system overview diagram along with descriptors) (Winikoff et al., 2001).

3. Describing the interactions between agents using interaction diagrams (developed from scenarios) and interaction protocols (developed from interaction diagrams) (Winikoff et al., 2001).

The detailed design phase: This looks at the internal structure of each agent and how it will accomplish its tasks within the overall system. The outcomes of this phase are detailed diagrams showing the internal functionality of each agent and its capabilities, process diagrams that show the internal processing of the agent, as well as descriptions of data structures used by the agent, plans and subtasks and the details of plan triggers (Bresciani et al., 2002).

Figure 1.1 indicates the main design artifacts that arise from each of these phases as well as some of the intermediary items and relationships between items. The figure shows the models and dependencies, but does not show the process (although it does imply it).

The development (and revision) of the various models depicted in Figure 1.1 is intended to proceed in an iterative fashion (similar to the Rational Unified Process) where in each iteration the focus of the work gradually shifts further down towards implementation. However, it is expected that most iterations will not be exclusively concerned with a single phase such that many iterations will involve revision of previously developed models(Winikoff et al., 2001)..

Furthermore, Figure 1.1 is divided horizontally and vertically. The three horizontal regions form the three phases of the methodology discussed above. The left-most region (consisting of scenarios, interaction diagrams, interaction protocols and process diagrams) deals with descriptions of the dynamic behaviour of the system. The middle vertical region (data coupling, acquaintance, system overview, agent overview and capability overview) deal with overviews of the system while the remaining models (the right region) give detailed descriptions for each entity in the system. Both

the middle and right region deal with the static structure of the system (Winikoff et al., 2001).

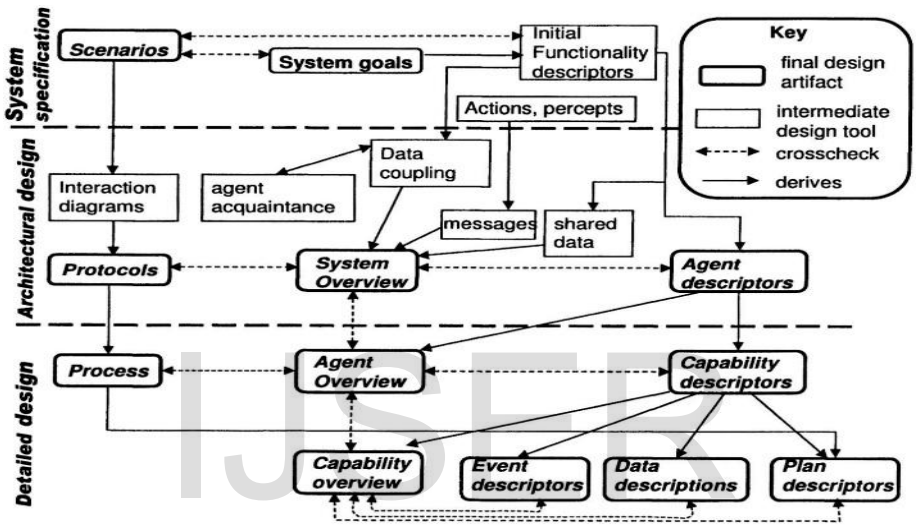

Figure 1.1.Overview of the Prometheus Methodology (Winikoff et al., 2001).

2.0 **Literature Review.**

Prometheus, like any other methodology, defines a number of system models and notations that are used to describe these models. We describe structural overviews at various levels (system, agent, capability) with a single diagram type. In addition, diagrams are used for showing data coupling and agent acquaintance relationships. Dynamic behaviour is currently described with existing models from UML (Unified Modeling Language) and AUML (Agent UML) (Winikoff et al., 2001).

In addition to graphical notations, we use structured textual descriptors (i.e., forms) for describing individual system entities (e.g., agents, functionalities, plans, etc.). We also maintain a data dictionary which is important in ensuring consistent use of names.

It is important to note that Prometheus is a general purpose methodology. In particular, most of the methodology (specification and architectural design) does not assume particular agent architecture. Although the detailed design phase does target a particular family of agent architectures (namely those that achieve goals using a library of plans), yet, did not make Prometheus special- purpose. Any methodology that addresses implementation needs to have a target platform. For example, Tropos (Bresciani et al., 2002) also targets BDI-like systems, whereas Gaia avoids the issue by not addressing implementation.

System specification consists of three main activities: determining the system's interface to the environment, determining the system's goals and functionalities, and determining scenarios which capture the usage of the system.

Since agents are situated, one of the key things to be captured in the development process is how the agents interact with their environment. For instance, following standard terminology as put forward by Russell and Norvig, (1995) we recall incoming information from the environment percepts and agents as a means of affecting the environment actions. As discussed in Winikoff et al., (2001) the raw data from percepts may need to be processed in order to obtain an event that are of significance event for the agent system. Prometheus prompts the developer to consider such issues such as a video frame from a camera on a soccer playing robot which may need processing to extract the symbolic objects, including ball, goal and players, as well as further processing to determine whether anything significant has actually happened, which may include a ball having moved since a previous frame, or a ball not appearing where one was expected (Winikoff et al., 2001)..

Furthermore, determining the system's goals and functionalities is done by iterating over the following steps (Russell and Norvig, 1995):

- Identify and refine system goals – main and subsidiary;

- Group goals into functionalities;
- Prepare functionality descriptors;
- Define use case scenarios (and variations); and
- Check that all goals are covered by scenarios.

An initial set of goals is identified from the initial requirements. These are refined and elaborated into a hierarchy of goals by asking how goals will be achieved, and why goals are being achieved (van Lamsweerde, 2001). For example, designing an online book store, we might have a high-level goal fully online system. This goal might have associated with the subgoals find books online, pay online and order online.

Functionalities are limited "chunks" of system behaviour that describe in a broad sense what the system needs to be able to do. We derive functionalities by grouping related goals. For example, given the goals above, we might also have another high-level goal of purchase books with subgoals, find books, place order, make payment, and arrange delivery. Pay online and make payment are clearly closely related if not identical goals, and are therefore grouped together in a single functionality (Russell and Norvig, 1995).

Functionality descriptors capture the name and description of each functionality as well as what events activate it; what goals it achieves; what actions it performs; what percepts it receives; what messages it sends/receives, and what data it uses and produces.

Use case scenarios are complementary to goals in that they show how processes are composed within the system. In developing goals, we are typically building up scenarios of how these goals will be part of various processes within the system. Scenarios enable us to specify some of this structure, which in turn may help to identify missing goals.

Furthermore, use case scenarios are also based on ideas from object oriented design but are more structured. This structure allows for automated cross checking, and automatic production of partial information for later design artifacts (e.g., protocols) (Winikoff et al., 2001).

The core of the use case scenario is the sequence of steps describing a particular example of the system in operation. Each step

can optionally have data read and data written noted as well as the functionality that performs that step. Each step can be a GOAL, ACTION, PERCEPT or SCENARIO, as well as OTHER allowing for additional step types, even though these cannot be used in automated processing. The following example illustrates the steps of a use case scenario in Prometheus (Winikoff et al., 2001).

Query Late Books Scenario

Trigger: User enquiry
GOAL: Determine delivery status
GOAL: Log delivery problem
ACTION: Request delivery tracking
GOAL: Inform customer
OTHER: Delay
PERCEPT: Tracking information received
GOAL: Arrange delivery
GOAL: Log books outgoing
GOAL: Inform customer
GOAL: Update delivery problem (Lin et al., 2001)

Functionality descriptors, goals, and use cases give different views of a common underlying system. Therefore, they should be checked for mutual consistency. For example an interaction between functionalities in a use case scenario should also be evident in the interactions field of a functionality descriptor. Again, each system goal should be represented in at least one scenario; use case scenarios should cover the important normal uses of the system as well as some error/unusual situations, in order to give an idea of how these will be handled (Winikoff et al., 2001).

.

3.0 **Methodology**

The Prometheus methodology supports the full life cycle, including testing and debugging. David Poutakidis,(2013) a research student of the authors(Lin Padghan and Michael Winikoff) , has been

working on debugging MAS, using design artifacts such as those produced by the Prometheus methodology.

One technique that we use to systematically examine the properties which lead to coupling and cohesion is the Data Coupling Diagram. Potential groupings are then evaluated and possibly refined using an Agent Acquaintance Diagram.

A data coupling diagram (Figure 1.2) consists of the functionalities and all identified data (not only persistent data, but also data the functionalities re-quire to fulfill their job). Directed links are then inserted between functionalities and data, where an arrow pointing towards the data indicates the data is produced or written by that functionality, whereas an arrow pointing towards the functionality indicates the data is used by the functionality. A double-headed arrow indicates that the functionality uses and produces the data. Edges between data and data or between functionality and functionality are incorrect syntax (and cannot be drawn in the tool) (Padghan and Winikoff, 2010).

The data coupling diagram is used to identify groupings which are linked by their data use. When assessing the diagram visually we are looking for clusters of functionalities around data. This is one important aspect in the analysis of potential groupings of functionalities. It is also used to guide refinements and changes to achieve a cleaner delineation between agents (Padghan and Winikoff, 2010).

Some reasons for grouping functionalities into a single agent are if the functionalities seem to be related or if they share a lot of information. Some reasons for *not* grouping functionalities are if the functionalities are clearly unrelated, or if they exist on different hardware platforms (Padghan and Winikoff, 2010).

.

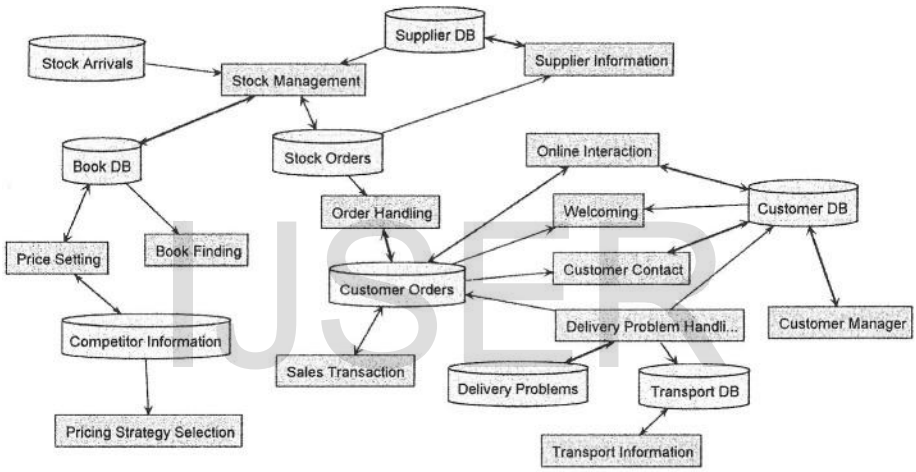

Figure 1.2. Data Coupling Diagram (Allan & Kurt, 2012).

In order to evaluate a potential grouping of functionalities into agents with respect to agent coupling, we use an *agent acquaintance diagram* (Figure 1.3). This diagram represents each of the agent types in the system. Information about agent interaction is extracted from the functionality descriptors and each agent type is linked with the other agent types it interacts with. Links can be decorated with the cardinality of the relationship if desired (e.g., one warehouse agent interacts with many sales agents) (Allan & Kurt, 2012). .

We then analyse the resulting diagram in two ways. One is simply an analysis of the density of the links within the diagram. It is a measure of the ratio of the actual coupling to the maximal possible coupling. If the system has four agents, then each agent could potentially be linked to a maximum of three other agents, giving a total number of 3 + 2 + 1 possible links. To get the link density we simply count the links and divide by this number. This measure is only one aspect of the analysis.We also consider bottlenecks and other issues(Avison, & Fitzgerald,2013)

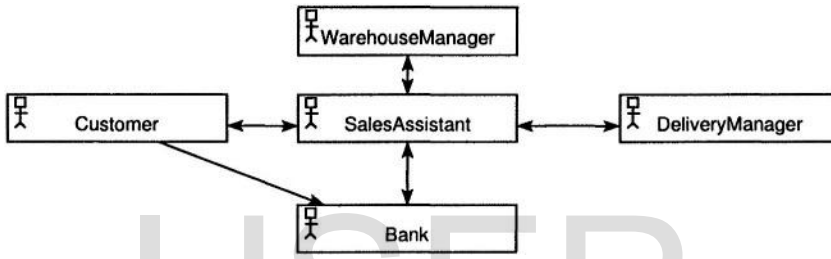

Figure 1.3. Agent Acquaintance Diagram (Avison, & Fitzgerald,2013)

4.0 System Design and Implementation

The system overview diagram is arguably the single most important artifact of the entire design process, although it cannot really be understood fully in isolation. The various descriptors provide the more detailed information that may be required.

The notation used in the system overview diagram (Figure 1.5) and in agent and capability overview diagrams is a directed graph where nodes represent design entities and directed arcs represent relationships. Figure 1.4 depicts the nodes that are currently used. These correspond directly to the concepts used in the Prometheus methodology.

A syntactically valid overview diagram consists of a set of nodes (excluding goals and functionalities), each labelled with a name, with links between them. We distinguish between “active” nodes (entities

that do things – agents, capabilities, and plans) and "passive" nodes (anything else – percepts, actions, messages, protocols, data). A link is valid from an active node to a passive node or from a passive node to an active node. A link is not valid from an active to an active node or from a passive to a passive node. An additional constraint is that there cannot be links to a percept and there cannot be links from an action.

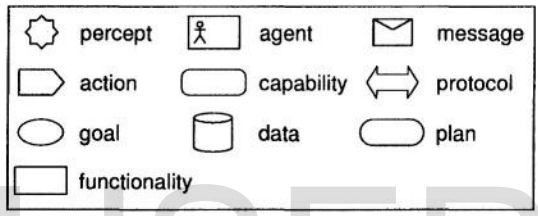

Figure 1.4. Notation used in Overview Diagrams (Burmeister & Cossentino, 2011).

The meaning of links is as follows (Burmeister & Cossentino, 2011):

- A link *to* a message indicates that the agent type (or capability or plan) sends that message.
- A link *to* a protocol indicates that the entity communicates using the protocol in question.
- A link *to* an action indicates that the entity performs the action.
- A link *to* a data node indicates that the entity writes to it.
- A link *from* a message indicates that the agent type (or capability or plan) receives that message.
- A link *from* a percept indicates that the entity receives the percept.

- A link *from* a data node indicates that the entity reads the data.

When drawing the system overview diagrams (Figure 1.5) we started by creating a named agent symbol for each agent type. We also add the percepts and actions at this point.

Furthermore, a data store icon is placed for each persistent data store, with an incoming link from each agent that writes to the data store and an outgoing link from the data store to each agent that directly accesses the data. Double headed links (arrows at both ends) indicate both read and write. Once interaction protocols have been defined they are added into the diagram and we indicate which agents participate in these protocols.

This sub-phase focuses on the system's dynamic behaviour by fully specifying the interaction between agents. Interaction diagrams borrowed from UML sequence diagrams are used as an initial representation of agent interaction. Fully specified interaction protocols (borrowed from the revised version of AUML currently under development) are the final design artifact (Avison, & Fitzgerald,2013)

Interaction diagrams are the same as sequence diagrams of UML except that they show interaction between agents rather than objects. One of the main processes for developing interaction diagrams is to take the use case scenarios developed in the specification phase and to build corresponding interaction diagrams, showing the interaction between agents in a scenario.

As with scenarios, we would expect only to have a representative set of interaction diagrams, not a complete set. In order to have complete and precisely defined interactions, we progress from interaction diagrams to protocols which define exactly which interaction sequences are valid within the system (Avison, & Fitzgerald,2013)

.

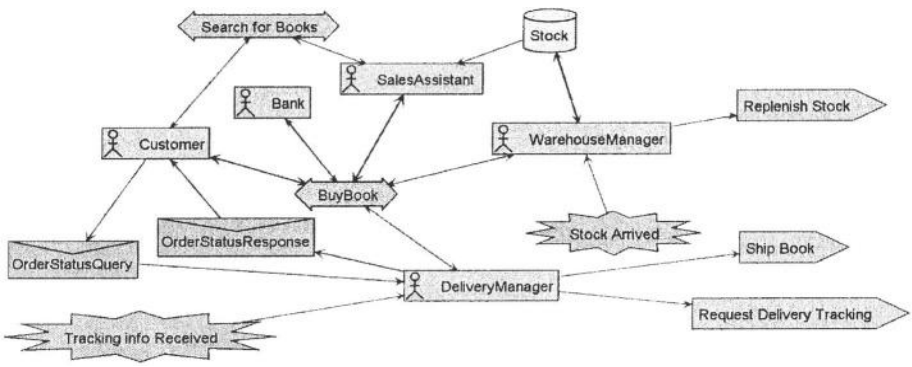

Figure 1.5. System Overview Diagram (excerpt) (Burmeister & Cossentino, 2011).

Developing protocols is done by considering alternatives. For each message (or percept) that an agent receives we ask "what are the possible messages that the agent could send as a response?" We then repeat the process for these messages. However, since protocols must show all variations, they are often larger than the corresponding interaction diagram and of need to be split into smaller chunks.

An example of interaction diagram is shown in Figure 1.6, while an example of interaction protocol (using the new AUML notation) is shown in Figure 1.7(Lin et al., 2001)

Furthermore, this phase deals with the internals of each agent, rather than the system as a whole. We use a hierarchical model so that each agent is broken up into capabilities. Capabilities may be included in more than one agent (Lin et al., 2001).

The steps with detailed design are:

Step 1: Develop agent overviews (showing interactions between capabilities) and capability descriptors. (Lin et al., 2001)

Step 2: Develop the internal process of an agent from the interaction protocols described, using a variant of UML activity diagrams. (Lin et al., 2001)

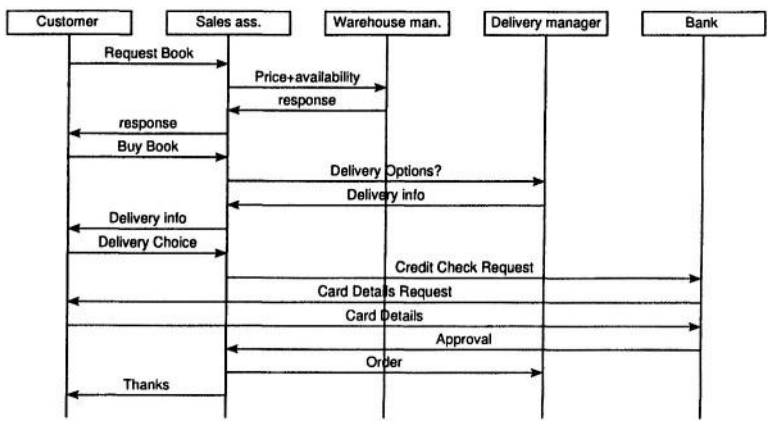

*Figure 1.6.*Interaction Diagram (Lin et al., 2001)

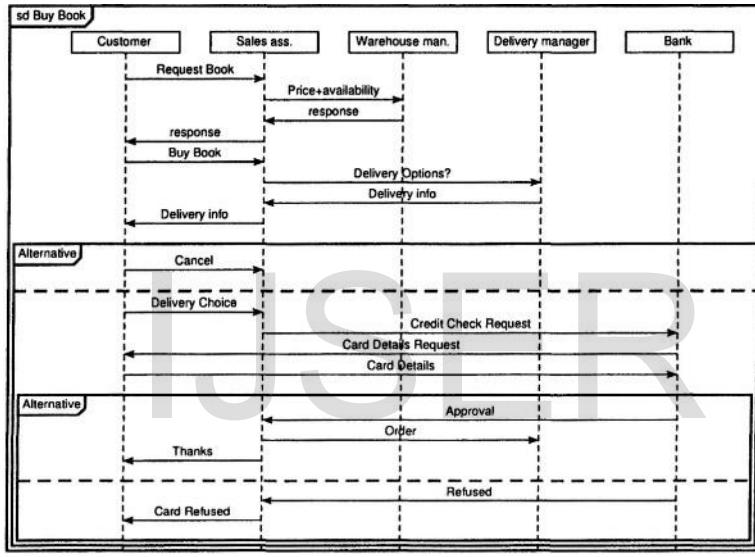

Figure 1.7.Interaction Protocol (Ardis, 2012).

Step 3: Develop the internal design of each capability in terms of plans, events, beliefs, and (possibly) sub-capabilities.

The process followed is essentially iterative refinement. We begin by considering for each agent what the agent needs to be able to do. Often, the functionalities that were grouped to form the agent type will be a good starting point for defining the capabilities of that agent type (Ardis, 2012).

We then connect up the capabilities. As depicted in the system overview diagram, each agent has incoming and outgoing messages, percepts that it received, actions that it performs, and data that is read and/or written. Each of these connections to an agent is mirrored in the agent overview diagram for that agent type. The agent overview diagram for a given agent type is quite similar to the system overview diagram, but shows interactions between capabilities within an agent, rather than between agents within a system. Any messages or percepts that are incoming to an agent in the system overview, must be incoming to some capability (or plan) within that agent in the capability overview diagram. Similarly any actions or messages that are outgoing from an agent in the system overview, must be outgoing from some capability (or plan) within that agent in the capability overview diagram.

Once capabilities (and plans) within an agent have been defined we consider each capability and refine its internals. This process continues until the internal operation of each agent and each capability is defined in terms of plans, messages, data, and other capabilities (Lin et al., 2001)

Designs for large systems are almost always developed incrementally with many revisions. When revising any artifact, be it documentation, code, or de-sign, it is easy to introduce inconsistencies and minor errors. We have found tool support to be extremely useful for checking and maintaining design consistency across varying levels of detail (Lin et al., 2001)

5.0 Discussion

The Prometheus Design Tool (PDT) allows a user to enter and edit a design, in terms of Prometheus concepts; check the design for a range of possible inconsistencies; and automatically generate a design report that includes descriptors for each design entity, a design dictionary, and the various diagrams. It also provides descriptor forms which prompt for the various aspects which should be considered. When any aspect of the design is modified, the change is propagated to all levels, although in some cases user input is still required for finalization.

PDT supports the Prometheus methodology in a number of ways. It supports the process of deriving agent types from functionalities by deriving part of each agent's interface, by cross checking the declared interface of an agent against the functionalities that make up the agent type. It also generates coupling and acquaintance diagrams. It supports the process of developing the internals of agents in the detailed design phase by cross

checking an agent's internals against the agent's declared interface, checking the consistency of a plan with its context, and by supporting views of design diagrams at different levels (system overview, agent overview, and capability overview). For more details on tool support for the Prometheus methodology, see Padgham and Winikoff, (2002).
The Prometheus Design Tool is currently available and further functionality is under development.

Specifically, the work described in Poutakidis et al., (2003); used interaction protocols expressed in AUML (Odell et al., 2000). These are translated into Petri nets and a debugging agent used these to monitor agent interactions and alert the programmer when a protocol is not followed correctly.

The latest version of the JACK Development Environment (JDE) includes a design tool that allows Prometheus overview diagrams (based on a slightly older version of the methodology) to be drawn. The JDE also includes a graphical user interface that allows the structure of an agent system to be built by drag-and-drop and by filling in forms.
The JDE supports the Prometheus methodology in that the concepts provided by JDE correspond to the artifacts developed in Prometheus' detailed design phase. It is important to realise that the agent structure described in the JDE generates JACK code that can be compiled and run. This automatic generation of skeleton code from design artifacts is extremely useful, and has encouraged students to do design prior to coding.

Conclusion.

Indeed, the Prometheus methodology has been developed over a number of years, as a response to both educational and industrial needs. During its development it has been used by industrial practitioners, taught at workshops at a number of conferences, and has been taught to undergraduate and postgraduate students, as well as having been used in student projects.

The Prometheus methodology has partially grown and assisted students in their efforts to develop agent systems, without a methodology, graduate student would flounder and end up building a system which made little real use of agents (Padgham and Winikoff, 2002).

The methodology has formed the basis for a course on agent-oriented design that is offered by Agent Oriented System (AOS) to industry software developers who are starting to use the JACK intelligent agents' development environment and has been successful in introducing them to methods to assist them in design of agent applications (Busetta et al., 1998). For example, a prototype weather alerting system developed for the Australian Bureau of Meteorology by Agent Oriented System (AOS) used Prometheus overview diagrams using the JDE to capture the design. The Prometheus overview diagram notation as implemented in the JDE was also used within Agent Oriented System (AOS) on a range of projects (Mathieson et al., 2004).

Again, the Prometheus methodology has also been taught to undergraduate students as a class. The class spends roughly half of the semester covering the methodology and the other half introducing the JACK agent programming language and plat-form. The students were able to design and implement reasonable agent systems in a single semester (Lin et al., 2001)

References

Allan, W., & Kurt, C. (2012). *A framework for evaluating software technology*. IEEE Computer Society Press.

Allan, M., & Wallnau, T. (2012). *The Rational Unified Process: An Introduction*. Addison- Wesley Pub Co.

Arazy, O., & Woo, C. (2012.). Analysis and design of agent-oriented information systems. *The Knowledge Engineering Review*, 17(2).

Ardis, A., John A., & James V. (2012). A framework for evaluating specification methods for reactive systems: Experience report. *In International Conference on Software Engineering,* pages 159-168.

Avison, D., & Fitzgerald, G. (2013). *Information Systems Development: Methodologies, Techniques and Tools.* McGraw-Hill: New York.

Avison, D., & Fitzgerald, G. (2011). *Information Systems Development: Agent Oriented Methodologies, Techniques and Tools.* McGraw-Hill: New York.

Barbara, A., Drogoul, A., & Benhamou, P. (2016). Agent-oriented design of a soccer robot team. In Proceedings of the Second International Conference on *Multi-Agent Systems*. Menlo Park, CA: American Association for Artificial Intelligence.

Barbara K., (2012). DESMET: a method for evaluating software engineering methods and tools. *Technical Report TR96-09*, University of Keele, U.K.

Bauer, B., & Odell, J. (2005). UML 2.0 and agents: how to build agent-based systems with the new UML standard, *Journal of Engineering Applications of Artificial Intelligence* Vol. 18, Issue 2, 2005.

Belina, F., Hofgrefe, D. & Sarma, A.,(2011). *SDL with Applications from Protocol Specification*", Prentice Hall Int., Hertfordshire, UK, 1991.

Behling, K., (2010). *Project Management – Theory and Practice*. McGraw-Hill professional.

Berard E. (2012). A comparison of object-oriented methodologies. *Technical report*, Object Agency Inc.

Bernon, C., & Glize, P. (2012). *The ADELFE methodology for an intranet system design*. Washington, DC: Cato Institute.

Berard E. (2011).*A comparison of object-oriented methodologies*, Technical report,Object agency Inc.

Bobkowska, A. (2005). *Framework for methodologies of visual modeling language evaluation*, Proceedings of the symposia on Metainformatics, ACM Press 2005.

Braubach, L., Pokahr, A., Moldt, D. and Lamersdorf W.(2005).*Goal representation for BDI agent systems*", In R. Bordini, M. Dastani,

J. Dix . and A. El Fallah Seghrouchni, editors, Programming Multi-Agent Systems, second Int. Workshop (ProMAS'04), vol. 3346 of LNAI, Pages 44–65. Springer Verlag,.

Brazier, F., Jonker, C. & Treur, J. (2010).*Principles of compositional multi-agent system development*", In Proceedings of Conference on Information Technology and Knowledge Systems, Pages 347–360. Austrian Computer Society.

Bresciani, P.,Giorgin,N., Hiunchiglia, R., Mylopoulos,K., & Perini, A. (2014).*Tropos: An agent- oriented software development methodology*. Autonomous Agents and Multi-Agent Systems.

Bresciani, P., & Perini, A. (2010).Tropos: *An agent-oriented software development methodology*. Autonomous Agents and Multi-Agent Systems.

Buhr, R.,(2008). *Use Case Maps as Architectural Entities for Complex Systems*, IEEE Transactions on Software Engineering. Vol. 24, No. 12, Pages 1131-1155, 2008.

Buhr, R., & Casselman, R.(2011).*Use Case Maps for Object-Oriented Systems*. Prentice- Hall, USA.

Burmeister, B.(2010). *Models and Methodology for Agent-Oriented Analysis and Design*; In Working Notes of the KI'96 Workshop on Agent-Oriented Programming and Distributed Systems, Saarbrilcken, Germany.

Booch, G. (2014). *Object-oriented analysis and design*. Redwood City, CA: The Benjamin/Cummings Publishing Company, Inc.

Booch, G. Rumbaugh, J & Jacobson.I. (1998). *The Unified Modeling Language User Guide*. Addison Wesley.

Burrafato, P. & Cossentino, M. (2012). *Designing a multi-agent solution for a bookstore with the PASSI methodology*. In P. Giorgini, Y. Lespérance, G. Wagner & E. Yu (Eds.), Proceedings of the Agent-Oriented Information Systems.

Burrafato, P., & Cossentino, M. (2012). Designing a multi-agent solution for a bookstore with the PASSI methodology. In P. Giorgini, Y. Lespérance, G. Wagner & E. Yu (Eds.), Proceedings of the *Agent-Oriented Information Systems.*

Burmeister, F., & Cossentino,M,.(2011). Designing a multi-agent solution for a bookstore with the PASSI methodology. In Fourth International Bi-Conference Workshop on *Agent-Oriented Information Systems* (AOIS-2011), Toronto (Ontario, Canada).

Bush,G., Stephen, C., & Martin P(2011). The Styx agent methodology. *The Information Science Discussion Paper Series 2012*, Department of Information Science, University of Otago, New Zealand.

Brain, J., & Odell, J.(2010.). *Object-oriented methods: Pragmatics and considerations.* Upper Saddle River, NJ: Prentice-Hall .

Caire, G., & Leal,F. (2012): Recommendations on supporting tools. *Technical Information Final version, European Institute for Research and Strategic Studies in Telecommunications (EURESCOM), July 2012.*

Caire, G., & Massonet, P. (2011). Agent-oriented analysis using MESSAGE/UML. In M. Wooldridge, G. Wei, & P. Ciancarini (Eds.), *Agent-oriented software engineering II.*

Castro, J., Kolp, M., & Mylopoulos, J.(2010). A requirements-driven development methodology. In In Proceedings of the 13th International Conference on *Advanced Information Systems Engineering (CAiSE'01)*, Interlaken, Switzerland.

Castro, J., Kolp, M., & Mylopoulos, J. (2012). Towards requirements-driven information systems engineering: *The Tropos project in Information Systems*.

Cavedon, L., & Sonenberg, L. (1998). On social commitment, roles and preferred goals. In Proceedings of the Third International Conference on *Multi-Agent Systems (ICMAS)*, Paris. IEEE Computer Society.

Cavedon, L., & Sonenberg, L. (2012). On social commitment, roles and preferred goals. In Proceedings of the Third International Conference on *Multi-Agent Systems (ICMAS)*, Paris. IEEE Computer Society.

Cernuzzi, L., & Rossi, G.(2012). *On the evaluation of agent oriented modeling methods*. In Proceedings of Agent Oriented Methodology Workshop, Seattle.

Cossentino, M., & Potts, C. (2012). A case tool supported methodology for the design of multi- agent systems. In The 2002 International Conference on *Software Engineering Research and Practice (SERP'02)*, Las Vegas (NV), USA.

Cernuzzi, L., & Rossi, G. (2002). On the evaluation of agent oriented methodologies. In Proceedings of OOPSLA 2002 Workshop on *Agent-Oriented Methodologies*. Sydney, AUS: Centre for Object Technology Applications and Research.

Cernuzzi, L., & Rossi, G. (2002). On the evaluation of agent oriented methodologies. In Proceedings of OOPSLA 2002 Workshop on *Agent-Oriented Methodologies*. Sydney, AUS: Centre for Object Technology Applications and Research.

Collinot, C., & Treur, J. (2010). Deliberate normative agents: Principles and architectures. In N. Jennings & Y. Lespérance (Eds.), *Intelligent agents VI Berlin*: Springer-Verlag.

Chan, K., & Karunasekera, S. (2004). Agent-oriented software analysis. In Proceedings of 2004 Australian *Software Engineering Conference* (pp. 20-27). Los Alamitos, CA: IEEE Computer Society Press.

Chris, S., & Barbara, A. (2011). Evaluating software engineering methods and tools-part 4: the influence of human factors. ACM SIGSOFT Software Engineering Notes.

David, L., & Michael W.(2011) . Debugging multi-agent systems using design artifacts: The case of interaction protocols. In Proceedings of the First International Joint Conference on *Autonomous Agents and Multi Agent Systems*.

David,L.(2012).*Agent-Oriented Information Systems*. Redwood City, CA: The Benjamin/Cummings Publishing Company, Inc.

Debenham, J., & Henderson-Sellers, B.(2010). Full lifecycle methodologies for agent-oriented systems the extended OPEN process framework. In Proceedings of *Agent-Oriented Information Systems*, Toronto.

DeLoach,S.(2012). Multiagent systems engineering: A methodology and language for designing agent systems. In *Agent-Oriented Information Systems '99 (AOIS'99)*, Seattle WA.

DeLoach, S. (2012). Analysis and design using MaSE and agentTool. In Proceedings of the 12th Midwest *Artificial Intelligence and Cognitive Science Conference* (MAICS 2001), 2012.

Dumke, R.(2011). Metrics-based evaluation of object-oriented software development methods. In Proceedings of the 2nd Euromicro Conference on *Software Maintenance and Reengineering (CSMR'98)*, pages 193{196, Florence, Italy.

E. Yu (2011). *Modelling Strategic Relationships for Process Reengineering*, University of Toronto, Department of Computer Science, 2011.

Eckert, G. (2010). *Improving object-oriented analysis*. Information and Software Technology.

Elammari, M., & Lalonde, W.(2010). "*An Agent-Oriented Methodology: High-Level and Intermediate Models*", *HLIM*, Proceedings of AOIS Heidelberg.

Frank, U. (2012). A comparison of two outstanding methodologies for object-oriented design. *Technical Report*.

Frank, U. (2012). Evaluating modeling languages: relevant issues, epistemological challenges and a preliminary research framework. *Technical Report* 15, 2012.

Genesereth, C., & Nelson, T. (2011). The importance of dealing with uncertainty in the evaluation of Software engineering methods and tools. In Proceedings of the 14th international Conference on *Software engineering and knowledge engineering*, ACM Press.

Glaser, C., & Francisco, L. (2010). Agent oriented analysis using MESSAGE/UML. In Michael W., Paolo, C., & Gerhard, W., editors, Second International Workshop on *Agent-Oriented Software Engineering* (AOSE-2012.

Hans –Van, V.(2012). A CASE tool supported methodology for the design of multi-agent systems. In H.R. Ababnia & Y. Mun (Eds.), Proceedings of the 2012 International Conference on Software Engineering Research and Practice (SERP'12), Las Vegas.

Hans-Van, V. (2000). *Software Engineering: Principles and Practice.* John Wiley & Sons, second edition.

IEEE Std 610.12. *IEEE Standard Glossary of Software Engineering Terminology*, p.77, 1990.

Iglesias,S.(2012). Evaluating modelling languages: relevant issues, epistemological challenges and a preliminary research framework. *Technical Report 15*, University of Koblenz-Landau.

James, O.(2012). Objects and agents compared. *Journal of Object Technology*. Toronto

Jayaratna,N.(2012). *Understanding and evaluating methodologies, NISAD: A systematic framework*", Maidenhead, UK: McGraw-Hill.

Jennings, N., Sycara, K., & Wooldridge, M.(2012). A Roadmap of Agent Research and Development; *In Autonomous Agents and Multi-Agent Systems Journal*, Publishers, Boston.

Jennings, N., & Wooldridge, M.(2012). *Agent-Oriented Software Engineering*, in proceedings of the 9th European Workshop on Modelling Autonomous Agents in a Multi-Agent World: Multi-Agent System Engineering (MAAMAW-99), vol. 1647, Springer-Verlag: Heidelberg, Germany.

Juan, T., Pierce, A., & Sterling, L.(2011). *Roadmap: Extending the gaia methodology for complex open systems*", In Proceedings of the 1st ACM Joint Conference on Autonomous Agents and Multi-Agent Systems (Bologna, Italy), ACM, New York.

Juan, T., Sterling, L. and Winikoff, M.(2002).*Assembling Agent-Oriented Software Engineering Methodologies from Features*", in the Proceedings of the Third International Workshop on Agent-Oriented Software Engineering, at AAMAS'02, Bologna, Italy, 2002.

Jefrey, M. (2011). An introduction to software agents. In Jefrey M. Bradshaw, editor, *Software Agents,* pages 3{46. AAAI Press / The MIT Press, 2011.

John, F. (2010). *Multi-agent Systems: An Introduction to Distributed Artificial Intelligence*. Addison-Wesley.

Kendall E.,(1996). *Agent Roles and Role Models: New Abstractions for Multi-agent System Analysis and Design*", Proceedings of the International Workshop on Intelligent Agents in Information and Process Management, Bremen, Germany, 1998.

Kendall, E., Malkoun, M., and Jiang, C. (2010).*A Methodology for Developing Agent Based Systems*", In Distributed Artificial Intelligence Architecture and Modeling, LNAI 1087. Springer-Verlag, Pages 85-99, Germany.

Khanh, H., & Michael W.(2010) . *Comparing agent-oriented methodologies*. In To appear at the International Bi-Conference Workshop on Agent-Oriented Information Systems (AOIS-2010), Melbourne, Australia.

Kruchten, P.(2000).*The Rational Unified Process: An Introduction.* Addison-Wesley Pub Co.

Krupansky, J. (2010).*Foundations of Software Agent Technology*", Agtivity: Advancing the Science of Software Agent Technology.

Kinny, T., Georgeff, B., & Rao, I.(1996). *Desire: Modeling Multi-Agent Systems in a Compositional Formal Framework,* Int.Journal of Cooperative Information Systems, Vol. 6. Special Issue on FormalMethods in Cooperative Information Systems: Multi-agent Systems.

Law,D., & Naem,A.(2013). *Methods for Comparing Methods: Techniques in Software Development.* NCC Publications

Lewis, R.(2014).Project Management. McGraw-Hill professional

Lin, P., & Michael, W., (2011). Prometheus: A methodology for developing intelligent agents. In Third International Workshop on *Agent-Oriented Software Engineering*, July 2011.

Lin, P., & Michael, W., (2011). Prometheus: A pragmatic methodology for engineering intelligent agents. In Proceedings of the OOPSLA 2002 Workshop on *Agent-Oriented Methodologies,* pages 97{108, Seattle, November 2011.

Lin, P., & Michael, W., (2011). Prometheus: Engineering intelligent agents. Tutorial notes, available from the authors, October 2011.

Lin, P., & Winkoff,A., (2012). *Prometheus: A brief summary*. Technical note.

Lin, P., & Michael W.(2012). *Prometheus: A methodology for developing intelligent agents.* McGraw-Hill: New York.

Lind, C., Erik, A., Steve J., Frank H., & Pascale, T. (2011). *Empirical studies of object- oriented artifacts, methods, and processes*: State of the art and future directions. Empirical Software Engineering.

Maes, J.(2011). *Understanding and Evaluating Methodologies: NIMSAD a Systematic Framework.* McGraw-Hill, New York, 2nd edition.

Mark, W., & DeLoach.S. (2012). An overview of the multiagent systems engineering methodology. In The First International Workshop on *Agent-Oriented Software Engineering,* Limerick, Ireland.

Michael, P. (2012). Evaluation of object-oriented modeling languages: A comparison between OML and UML. In Martin Schader and Axel Korthaus, editors, *The Unified Modeling Language Technical Aspects and Applications*,. Physica-Verlag, Heidelberg.

Mike, F. (2012). The Tropos software development methodology: Processes, Models and Diagrams. In Third International Workshop on *Agent-Oriented Software Engineering*.

Moulin, A.(1994). Multiagent systems engineering. *International Journal of Software Engineering and Knowledge Engineering*.

Nicholas, A., & Wooldridge, M.(2012). *An Introduction to Multi-Agent Systems*. John Wiley & Sons, 2002.

Nwana, H. (2012). *Software agents: An overview*. Knowledge Engineering Review. Toronto

Odell, S. (2012). Requirements of an object-oriented design method. *Software Engineering Journal*, pages 102-113, March 2012.

O'Malley, S.(2011). Determining when to use an agent-oriented software engineering methodology. In Proceedings of the Second International Workshop On *Agent- Oriented Software Engineering (AOSE-2011).*

Omicini, J., Parunak H., & Bauer B. (2012). *Representing Agent Interaction Protocols in UML*. The First International Workshop on Agent-Oriented Software Engineering.

Padgham, .J., & Winikoff, A. (2012). *A cognitive foundation for comparing object-oriented analysis methods*. In J. F Nunamaker and IEEE Computer Society Press: R. H Sprague, editors, 26th Hawaii International Conference on System Sciences.

Padgham, J., & Michael, I.(2012). *Agent Oriented Methodology;* Addison Wesley.

Parson, C., Jazayeri, M., Mandrioli, D.(2011): *Fundamentals of Software Engineering*. PrenticeHall, Englewood Cliffs, N. J. ,pp.12,14.

Pollack, B., Fausto, M., & Anna P. (2010). *Troops: An agent-oriented software development methodology*. Technical Report DIT-02-0015, University of Trento, Department of Information and Communication Technology.

Rao, A.(2010). *A methodology and modeling technique for systems of BDI agents*, In Proceedings of the Seventh European Workshop on Modelling Autonomous Agents in a Multi- Agent World, LNAI Vol. 1038, Springer-Verlag, Berlin.

Roel, W. (2011). *A survey of structured and object-oriented software specification methods and techniques.* ACM Computing Surveys.

Rumbaugh, E.(2011). A comparison of object-oriented methodologies. *Technical report*, Object Agency Inc.

Russel,M., & Norvig.(2011). Determining when to use an agent-oriented software engineering methodology. McGraw-Hill, New York.

Sabas,M., Badri,O., & Delisle,M.(2002). *The Gaia methodology for agent-oriented analysis and design.* Autonomous Agents and Multi-Agent Systems.

Scott, A.(2011). Specifying agent behavior as concurrent tasks: Designing the behavior of social agents. In Proceedings of the Fifth Annual Conference on *Autonomous Agents*, Montreal Canada.

Scott, A. (2012). *Applying agent oriented software engineering to cooperative robotics*. University of Toronto, Department of Computer Science.

Scott, A. (2012). Multi agent Systems Engineering. *International Journal of Software Engineering and Knowledge Engineering.*

Sharble, R., & Cohen,S.(2012). The object-oriented brewery: *A comparison of two object oriented development methods*. SIGSOFT Software Engineering Notes.

Sturm, A., & Shehory,O. (2003). Towards industrially applicable modeling technique for agent- based systems (poster). In Proceedings of International Conference on *Autonomous Agents and Multi-Agent Systems,* Bologna.

Tambe, M., & Jennings, R.(2012). *Applications of intelligent agents. Agent Technology:* Foundations, Applications, and Markets

Trans,O., & Law,M.(2010). *Understanding and Evaluating Methodologies:* McGraw- Hill, New York.

Verharen, G., & Dignum, A. (2012). Agent-Object-Relationship Modeling. In Proc. of Second International Symposium - from *Agent Theory to Agent Implementation together with EMCRS 2012,* April 2012.

Wagner, G., (2011). Agent-Oriented Analysis and Design of Organizational Information Systems. In Proc. of Fourth IEEE International Baltic Workshop on *Databases and Information Systems*, Vilnius (Lithuania), May 2011.

.

.

.