A Conceptual Agent Cooperation Model for Multi-agent Systems’

Team Formation Process

Ebrahim AlHashel
Faculty of Information Science and Engineering
University of Canberra - Australia
Ebrahim.al.hashe@canberra.edu.au

Abstract

Agent cooperation is the key advantageous feature of agent-oriented software (AOS) in comparison with all other software approaches. Agent cooperation process has the potential to dynamically form a team of agents, and then execute a plan to achieve a common goal. Applying the agent cooperation concept requires two software engineering tools: the first is the software development methodology that can design the cooperative agent-based system; the second is the agent cooperation model that can manage the team behaviour at runtime.

This on-going research attempts to engineer the agent cooperation concept into two steps: the first step, the Prometheus development methodology is enhanced to acquire the capabilities to generate the agent team artefacts that will be consumed by the proposed cooperation model; in the second step, a conceptual cooperation model is developed. In regard to the complexity of agent cooperation process, the implementation part of the cooperation model held for further investigated.

Keywords: Software engineering, agent-oriented software development, agent-oriented methodology, Multi-agent system.

1. Introduction

Designing a team of agents that can work together toward a common goal is one of the challenges in the research area of agent-oriented software engineering. There are many theories and attempts directed to formulate the team formation process or what is known in the agent-oriented field by agent cooperation [1], [2], [3], [4], [5], [6], [7]. The existing agent-oriented software development methodologies e.g. Prometheus [8], Gaia [9], AUML [10], MaSE [11], PASSI [12], Tropos [13], directed to engineer multi-agent systems that can practise the cooperation concept which means forming teams of agent dynamically. In addition the available well known agent platforms, namely JADE [14], Zeus [15], RETSINA [16], IMPACT [17], JACK [18], and Aglet [19], are designed toward variant agent services and approaches other than dynamic team formation process. This limitation is situated at the core of agent-oriented software engineering practice and is reflected negatively on the agent-base applications. As a result, the multi-agent systems can not function efficiently or exercise its powerful role.

Developing agent cooperation activity requires two main components: first, there must be an agent software engineering development methodology that has the potential to design and deliver the elements that are needed to form the agent team process. The second component is the agent cooperation model (ACM) that works as a runtime environment to control and coordinate the operation of the team formation process. In fact ACM will manipulate the cooperation activities and provide the techniques to operate and integrate the entire cooperation process.

Considering the effort required to develops the overall project and the complexity of the agent cooperation process, this research partitions the project development plan into three phases. Phase one develops the software engineering development methodology and then designs the structure of the proposed agent cooperation model. Phase two will concentrate on the implementation and testing activities of the logic required to develop the cooperation model. Phase three will deal with the user interface, and determine how to interpret the user objectives and convert them into goals.
This research paper focuses on phase one only and divides the phase into two tasks. Task one is developing the software engineering development methodology that can engineer the agent cooperation process. Task two is to design the conceptual structure of the agent cooperation model that works as a runtime environment for the cooperation process. The development methodology will deliver the main elements required by the agent cooperation model in order to build its coordination and execution plan. Figure 1 shows an overview of phase one, focusing on the relationship between the modified development methodology (Prometheus) and the proposed agent cooperation model.

![Phase one overview diagram](image)

Figure 1. Phase one overview diagram.

Rather than reinventing the wheel by developing a new development methodology, Prometheus will be enhanced for the purposes of this research. Prometheus was selected among the other development methodologies for its originality and its ability to design individual BDI agents, Prometheus is supported by a development tool, and its artefacts are mapped seamlessly into the implementation.

This paper is organised as follows: Section 2 demonstrates agent cooperation concept and then discusses the well known theories that focus on formulating the cooperation concept. Section 3 explores Prometheus development methodology focusing on the detail design phase to illustrate the merging process between the current structures and the cooperation model descriptors. Section 4 overviews the proposed cooperation model, explaining the functionality of the main components. Section 5 describes how to incorporate the cooperation model elements with Prometheus analysis and design processes. Finally, the paper concludes with section 6 which summarises the research and then discusses the future work.

2. Multi-agent Systems Classification

The word cooperation is related to human behaviour. In the dictionary cooperation is defined as “work jointly toward the same end”. According to Luck et al. [3], there is a difference between cooperation and engagement. In cooperation, the participant must be autonomous (possess its own decisions), while in engagement there is no choice.

In the agent cooperation Smith and Davis [20], described agent cooperation as a pre-designed role within each agent logics to establish goals adaptation phenomena between agents, provided that there are no resource conflicts. However, the research defines agent cooperation as “the process of dynamically forming a team of agents toward the achievement of common goal”

Agent-oriented systems are categorised by the levels of cooperation (team work) that take place between the agents of one system forming a Multi-agents system. Cooperation is the central behaviour of multi-agent systems and the type of cooperation determines the type of multi-agent system. Figure 2 shows the multi-agent system typology based on the panel discussion at the First Workshop on Foundations of Multi-Agent Systems committee.

![Cooperation typology](image)

Figure 2. Cooperation typology [2].

Multi-agent systems (MaS) are classified as independent, if the system does not include cooperation between their agents. They are classified as discrete if their agents having their own objective isolated from others. Multi-agents system is cooperative if there is cooperation between their agents. There are two types of non-communicative cooperative systems where agents adjust their behaviour depending on non-messages or protocol sensors, for example, Observes and Act (OA) technique. The other type of cooperative system is the communication based system. These systems can be deliberative, (plan their actions) or negotiative which is both deliberative and competitive.

2.1. Agent Cooperation Theories

There have been many attempts to formulate agent social behaviour “cooperation”. Those attempts have
resulted in different efforts, theories, research reports, conference papers, and so on. The study of those attempts centres on the **team formation process** of agents in achieving a common goal [21], [22], [23].

The team formation process has been designed in different approaches, including, agent motivation, execution plan, organization structure, built-in objective. For example, Joint Intention introduces shared beliefs. Another notion, Shared Plan, is based on sharing the execution plan. Planned Team Activity is based on individual BDI and predefined plans within agent internal states. Wooldridge and Dunne [24], also presents a model based on the desire to achieve one of a set of goals. This set of goals is linked with the coalition choice then this choice leads to corresponding cooperation. There are also some attempts which rely totally on the capability of agent interactions using standard communication protocols. For example, Vieira et al. [25], has been developed knowledge based semantic to be incorporated into agents programming language known as AgentSpeak [26]. This semantic expand AgentSpeak logic to recognise, agent communication messages and transform them into knowledge and subsequent action related to this knowledge.

There are also other collaboration models, ideas, arguments, and contributions which have their own views on the process. To be more specific, cooperation is an open debatable subject [2]. For example, what is initiate cooperation, and what is the cooperation structure is still a research questions without definite answer.

With respect to all the existing cooperation concepts, they do not easily lend themselves to, for example, complexity-theoretic analysis and this is why efficient multi-agent applications are incomplete. Furthermore the existing ideas of team formation centred around distributed problem solving using multi-agents system approach. However, the research aims around designing a cooperative system from user perspectives. The cooperative system should be able to operate in open heterogeneous distributed environment, similar to the web-services to satisfy the user purposes.

### 3. Prometheus Overview

Prometheus [8], consists of three phases, as shown in figure 3: system specification, architectural design, and detailed design. The first phase, system specification, deals with determining the system’s environment and establishing its goals and functionalities. The environment is defined in terms of percept and actions while the system’s functionality is identified in terms of goals and plans. The outcome of this phase are system goals, scenario and functionality descriptions. The second phase, architecture design, uses and processes the system specification artefacts to develop a high–level designed agent system. The outcomes of this phase are agent types that will be used by the application, the interactions between agents, and overall system structure. The third phase, detailed design, will produce three main artefacts: agent capabilities, plans, and events. The implementation process is kept open for the developer to choose a preferred platform. But the use of the JACK development environment is strongly recommended because both Prometheus and JACK are oriented toward BDI agent. Prometheus development tool PDT provided an option to the developer to automatically generate JACK skeleton java code.

![Figure 3. The phases of the Prometheus methodology.](image)

**3.1. System Specification Phase**

The system specification phase focus on the analysis techniques. For the purpose of clarity, some low level scenario details are ignored in this description to instead concentrate on the big picture. The system specification phase focuses on system requirement definition to capture the system goals and sub-goals. The system goals are the centre construct of the system specification and are fundamental to AOS. The goals are systematically captured by searching for intentional words in the initial system documents. Once the main goals are identified, the other goals and sub-goals can emerge by using, refinement techniques (asking “how?” and “why?”) [27].

**3.2. Architecture Design Phase**

The architecture design phase will use the system specification artefacts to build the system architecture. The system architecture will be developed in three main steps. In the first step, the application agent types are specified; in the second step, the system interactions are
specified, in the third step, the system overviews are designed.

**Specification of Agent Types:** The objective of this step is to identify the types of agents embraced by the required application. During step one, agent descriptors will also be developed. This objective will be reached through the implementation of low coupling and highly cohesion principals in the diagram of system functionalities. Figure 4 represents the data coupling diagram after compilation of functionality descriptors.

![Figure 4. Example for initial data coupling diagram.](image1)

The techniques are centre around the relationships between the functionalities and data related to these functionalities. First, the data coupling diagram is built then the mostly related functionality is grouped in respect to the data relationship. The simple, concept to follow in grouping is to find if the group elements are associated with each other. Then the situation is read from a cohesiveness perspective to determine if a simple name (agent name) can be given. Identifying agents are demonstrated by figure 5, compiled by the data storage relationship to each cluster.

![Figure 5. Functionalities grouping diagram.](image2)

To review the agent coupling, the agent acquaintance diagram is developed. The acquaintance diagram will validate the type of agents involved in the application, and ensure that there is a loosely coupling association between the agents. The last step in the agent type specification is to print the agent descriptors.

**Developing System Interactions:** Specifying the system interaction (interactions between agents) is the second step of the architecture design phase. The purpose of this step is to capture the dynamic aspects of the system, by developing interaction diagrams from a scenario, generalizing interaction diagrams to interaction protocols then developing protocol and messages descriptors. Prometheus Development Tool (PDT) has the capability to compile the protocol scripts allocated in the system overview design and then generate the protocol diagram automatically.

### 3.3. Detailed Design Phase

Prometheus phases are integrated with each other. The detailed design phase uses the system architecture artefacts and develops two main aspects: first the internal individual agent capabilities and process; second the capabilities, plan and events analysis.

**Capabilities:** The first step in the detailed design is to compile the agent descriptor and agent capabilities to develop the agent overview diagram. Figure 6 presents the agent overview diagram developed as an example using PDT detailed design process. PDT has the ability to validate each design entity dynamically while the development process is running.

![Figure 6. Capability diagram developed using PDT.](image3)

**Process Specification:** The next step in building the individual agent is to identify the internal process specification of a single agent and specify its activities structure. Prometheus does borrow agent UML (AUML) activity diagram notation to present the process specifications. This means PDT does not provide any support to the process diagram. However, the best method to start building this task is to look into the protocols involved in the agent structure, the scenarios developed and the goals of the agent.

**Capabilities Overview Diagram:** The last milestone in the detailed design phase is to turn the analysis artefacts into the implementation platform and achieve Prometheus agent architecture, which is based on the concept Belief-Desire-Intention (BDI). This will
enforce the analysis process to focus on the internal structure of the system capabilities. Each capability is then decomposed into its lower level and the set of plans that provide the details of how to achieve the goals retrieved. Figure 7 presents an example for the capability overview diagram.

![Figure 7. Capability overview diagram.](image)

3.4. Implementation Phase

Prometheus has no restrictions for the implementation to any development platform. The decision to select which platform, depend on the nature of the application. But because Prometheus design BDI agent, its artefacts mapped seamlessly into JACK agent platform, see figure 8. This mapping is available within the PDT tool menu as an option. This option can compile the system diagrams and generate JACK Agent Language skeleton codes. The generated skeleton code will be used a starting point to develop the agent classes using java development language.

<table>
<thead>
<tr>
<th>Prometheus Detailed design artefacts</th>
<th>JACK classes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Agent</td>
<td>Agent</td>
</tr>
<tr>
<td>Capability</td>
<td>Capability</td>
</tr>
<tr>
<td>Plan</td>
<td>Plan</td>
</tr>
<tr>
<td>Goal</td>
<td></td>
</tr>
<tr>
<td>Data</td>
<td>Beliefset</td>
</tr>
<tr>
<td>Messages</td>
<td>Event</td>
</tr>
<tr>
<td>Percept</td>
<td></td>
</tr>
<tr>
<td>Action</td>
<td></td>
</tr>
</tbody>
</table>

![Figure 8. Mapping from Prometheus artefacts to JACK semantics.](image)

3.5. Prometheus limitations

Prometheus successfully analyses and designs an agent-oriented system. It has the capacities and techniques to design internal agent structures, including agent interactions. The artefacts used by Prometheus methodology are generally compatible with agent-oriented software concepts and those artefacts are seamlessly transferred to implementations. Our experience with Prometheus development methodology is that it is an excellent initiation in the field of AOSE, but still needs further improvements. In fact, Prometheus designs a system consisting of independent agents wrapped by interaction protocols. It does not contain any technique to deal with a system that consists of multi-agents working together to achieve a common goal (agent cooperation) [8].

To promote Prometheus further, we propose some points that need technical improvement. Although Prometheus limitations are beyond the scope of this research paper, but for the interest of documentation, they are as follow:

1. There is no requirements analysis phase.
2. Data set needs to be defined, analysed, and designed within the methodology process.
3. Capabilities and plans require further analysis levels to clarify the process.
4. Alternative plans need further analysis and refinements.
5. The context condition mechanism needs to be improved to provide more details and to easily map the agent belief to the alternative plans. This will create an association between the data structure and the system plans.
6. Prometheus does not design agent cooperation (agent team work) to achieve a common goal (aim of this research).

4. Proposed Agent Cooperation Model

The proposed cooperation model has been designed with the consideration to the following criteria:

1. Doable with the available software technologies.
2. Realistic with the vision that what we are designing are computer software programs.
3. Apply this research originated agent cooperation concept based on professions and skills with consideration of the Share Plan Theory (SPT).

The architecture of the proposed agent cooperation model is inspired by the definition “agent-oriented system is a cooperative type of system that has the ability to operate in open distributed heterogenic environment similar to the Internet environment, working on behalf of the user to achieve his objective”. This definition enforces three main inputs into the design process:

1. The user has an objective and needs the agent system to perform his objective with minimum user interference (user base cooperative system).
2. The system has the ability to interact with the web services, then associates a level of integrity with the web-services functionality to satisfy the
user objective (integrated with the web-services functionality).

3. The system operates in an open dynamic environment. For this reason the system will be able to dynamically establish task domains encompassing professions and skills (ability to establish task domain).

4.1 Cooperation Model Architecture

The architecture of the proposed cooperation model in figure 9 consists of four levels: user level, coordination level, professional level, and skill level. The user level is characterised by the user agent located in the early stage to communicate with the user and understand the users’ goals (objectives). The user agent is the application interface that guides the user in the definition of his objective (what he wants the system to do for him) and the coordination of this objective with the system capabilities (what the system can do). In this situation, the user agent will not accept objectives beyond the system capabilities. When the objective is found to be valid, then the user agent will start to communicate with the coordination level.

![Figure 9. Proposes agent cooperation model (ACM).](image)

The coordination level consists of two main agents: the coordination agent and the bidding agent. The coordination agent communicates with the user agent and the bidding agent. When the coordination agent receives the user objective from the user agent, it will convert it into a set of goals, then send it to the bidding agent. The coordination agent must also act as coordinator between the user agent and the bidding agent in terms of understanding both sides’ configurations, to map the user agent request with the bidding agent results. The bidding agent will receive the goals from the coordination agent, then it will advertise the goals on the blackboard in a format understandable to the professional agents. The bidding agent is also responsible for controlling the service traffic, for example, when to advertise, and when to receive goals and results.

The professional level is a container for a multiple number of agents; each has its own profession. The profession agent is categorised by the ability to perform in a specific area of a specialist domain (goal), for example, car hire, flight reservation, hotel booking. Each professional agent has its own expert domain supported by an unlinked set of integrated agents (skill agents) necessary to accomplish the selected goal. The professional agent is responsible to check the bidding blackboard agent and pick a goal within its profession domain. The professional agent has the goal procedure directory (GPD) that provides information about each goal requirement, including agent skills (what it can do), its domain addresses, and communication channels. Using the goal procedure directory the professional agent can form a goal domain, consisting of skills agents to perform the user goal or, form a team of agents to achieve the user goal.

The skill level consists of a set of agents attached by built in units of skills that are suitable to achieve a particular task or sub-goal. The sub-goal will be supplied by the professional agent to the skill agent. The skill agent will process the task according to the task sequence coming from the GPD. At the skill level, it is allowed to create a tree of skill agents related to each other working in task and sub-task relationships. The length of the tree is subject to the nature of the application and system designer decision. There are interaction activities between the skill agents in multiple directions parent to parent, parent to child, child to child of the same tree. In the tree architecture, the parent agents must have the task procedure directory (TPD) which contains the information about how the task can be achieved, including the sequence, resources, and agents that can achieve the task.

When the sub-goal has been achieved by the skill agent, it will be sent to where it comes from, either to the parent skill agent or to the professional agent itself, depending on the skill tree structure, until it reaches the node (professional agent). The professional agent will reassemble the result, if decomposed, and send it to the blackboard agent. The blackboard agent will match up the goal with the advertising method and start to reassemble the goal, if decomposed, then send it to the coordination agent. The coordination agent will reassemble the goal equivalent to the user objective and send it to the user agent. The user agent will validate the result and interact with the user, then display the final result.
5. Incorporating the Cooperation Model

The components of the proposed cooperation model are correlated to each other but not necessary to implement the idea as one body, so we can see each level as an independent plug-in component. Therefore, the user level will be allocated in the requirement phase. The coordination level will be merged with the specification phase where the professional and skill level will be implemented in the detailed design phase.

User Level: The scenario approach is an effective method to develop the user-goal requirement and can be used straightforwardly. The first main scenario will be retrieved from the system specification document by asking what the user wants the system to do on his behalf. After developing the first level scenario, the second refinement starts by finding what happens if the first level fails. This process continues until the exact user goal requirements are captured. The idea is like reversing the use-cases model in UML. Instead of identifying the actor to system functionality, it is identifying the system to user requirements. The following steps represent the approach:

1. Read system specification document and describe the typical interaction between the user and the system.
2. Develop the high level scenario.
3. Refine the high-level scenario to capture the user’s goal.
4. Use the user-goal diagram (see figure 10).
5. Develop the goal descriptor (see figure 11) for each user goal.

![User-Goal Descriptor](image)

**Figure 10. Example for user-goals diagram.**

![User-Goal Descriptor](image)

**Figure 11. User-goal descriptor.**

The main difference between percept and user-goal is that the percept is linked to a particular action or event in the system, whereas the user-goal can be linked to many precepts. In fact, the user-goal is at a higher level than the percept.

The Coordination level: The blackboard agent works in two directions: first as advertisement services, second as goal handler. It is a logical unit in the cooperation model, but at this stage of detailed design, we would like to define the elements that will drive the process, for example, professional agent capabilities, goal execution, and protocols. Figure 12 depicts the goal execution plan descriptor that will provide information to the blackboard about the professional level agent’s capabilities and responsibilities. The codes in the goal execution plan descriptor are related to the main descriptors of each activity where the name of each activity is presented for the consideration to maintain Prometheus standard.

![Goal Execution Plan](image)

**Figure 12. Example of goal execution plan descriptor.**

The professional level: The professional level consists of high level agents that own the strategy of executing particular tasks using a set of skills agents. Each professional agent is loaded with the one execution plan corresponding to its objective, including its relevant skill agent’s team, capabilities, and communication protocols. The agent designed by Prometheus will be linked to this agent level and contains additional information related to the logical professional level platform. Figure 13 and figure 14 depict the two independent descriptors that will be developed in the detailed design phase, in sequence with capabilities, plans, and events.
The Skill Level: The skill level is the factory that the system depends on. At this level all the skill agents are located in a tree configuration, as described earlier. Any system must contain a skill agent regardless how small it is, but a sub-skill level does not necessarily exist. This depends on the size and the design of the application. The agents that design through Prometheus development methodology can be directly attached to the skill level along with its enhancement descriptor. The additional descriptor elements will enable the skill agent to coordinate and communicate with the professional level and with each other at the same time. Figure 15 and 16 show how the current agent descriptors have been redesigned.

The Sub-Skill level: The sub-skill level will emerge if the system designer finds it suitable for the system. It leads to an efficient design; however, efficiency depends on the nature and the size of the system. If there are any needs to extend to this stage, then the relationship between the skill level and sub-skill level takes the same structure as that of the professional to skill levels. Regardless of the number of nodes in the skill trees, the agent in the node acquires the professional agent role and the descriptors will also flow in parent child fashion.

6. Conclusions and Future Work

The Agent cooperation concept is a powerful tool situated in the core of agent-oriented software engineering. Agent-oriented system without cooperation behaviour is classified as discrete and has no advantage over object-oriented software. Building agent cooperation concept is a difficult task and needs further research effort. However, taking the first step toward agent cooperation modelling enhances the agent-oriented software engineering.

The architecture of the agent cooperation model (ACM) is set up based on the matching between the goal and the professions and skills the team possess. The research found that one agent will cooperate with other agents to achieve a common goal. This needs a mapping between the agents’ capabilities and the shared goal. Agent abilities are identified by the profession that agent possesses. The profession is supported by a set of skills agents.

The research has enhanced Prometheus in different areas and made it integrated with the ACM and supply the elements that are essential to the cooperation process.

The second phase of this research is dedicated toward the implementation and testing of the agent cooperation model. The result will be reported in the next research paper.
Reference


[16] RETSINA, "development toolkit http://www.ri.cmu.edu/projects/project_76.html".


