A Formal Architectural Style for Designing Multi-agent Systems

Amir Anvarian Fard and Vahid Rafe

Abstract

Architectural styles include specific features which restrict the activities scope, an inevitable fact in software engineering. The appropriate selection of an architectural style is essential in the implementation phases. The attempt is made here to integrate the architecture concept with artificial intelligence, in specific its multi-agent system. The attempt is made here to realize the Artificial Intelligence system’s concept, the agent oriented systems in specific and assess the style concept and its implementation in multi-agent systems. By assessing the available styles and their drawbacks a new multi-purpose style is introduced for agent orientation. This style consists of four style concepts and analyzes all aspects of agent orientated system. This style is modeled through graph transformation system in order to be conceivable dynamically and be checked by a tool named GROOVE which automatically confirms lack of errors in this proposed style.

Keywords: Agent Oriented, Formal Method, Graph Transportation System, GROOVE Software, Software Architecture Style, Model Checking, Multi-agent

1. Introduction

Nowadays with the advances made in Artificial Intelligence and its vast application in industrial and scientific fields, the agent oriented systems have developed as main part of Artificial Intelligence. Recently agent oriented systems both in the software aspect for instance in the industrial controlling systems and in the hardware aspect like Extinguisher robots have shown considerable applications. In multi agent environments, agents operate autonomously but they must become synchronized with other agents. Existence of a central synchronizer that can save the agents autonomy is essential. Since for implementation of any software the selection and application of software architecture style is considered as an indispensable part of it, and that the agent oriented systems are considered as a software system, for multi-agent applications just like other software engineering issues different architectural styles are presented.

The greatest drawback in the represented styles up to now is that each one of these styles has viewed the agent oriented concept in a single context. Some of these styles have dealt with the issue in the distribution and communication bottlenecks sense. Some have considered the agent as a single factor with ignoring its connection with other factors; and yet some have considered its organizational theory. Another drawback of the available styles is that the basic concepts of the existing architectural style such as components, connectors and configurations rules are not clearly tangible. Moreover there exists no software architectural style which is modeled through a formal method in the artificial intelligence domain and thus the correctness of the presented style in architectural level cannot be verified automatically.

As the attempt is made here to present an architectural style where different viewpoints such as distribution way, organizational theory and internal structure of an agent assess the agent oriented system in a sense that the constituent parts of architectural style such as components, connectors and configuration rules are clearly defined and obvious. This style is going to be modeled through graph transformation systems which are formal methods.
2. Related Work

The concept of Artificial Intelligence was introduced through Touring test in 1950s\(^1\) at the same time when computers evolved at the 1950s. As the computer science expanded the field of Artificial Intelligence developed in various branches such as soft computing\(^2\), expert systems (the rule systems) and agent oriented systems. Other conceptions of Artificial Intelligence is mostly manifested in the agent conception which can exist in both hardware and software. One of the unique features of the agent is its autonomous nature in a sense that it realizes its environment-conditions independently as well as react on its own. As the concepts of software engineering shaped, the concepts like software architecture began to shape up. Architectural style refers to a set of architectural designing decision in specific area where the system is divided into set of components and connectors by defining the configuration rules between the components\(^4\). As software engineering advanced its concepts should have been clarified in a scientific manner in Artificial Intelligence fields and agent oriented systems. Nevertheless, some methodologies like Tropos\(^5\), Mase and Prometheus\(^6\) are presented for agent oriented systems’ production circle. In these metrologies stages like providing requirement, analyzing, designing and implementation were inserted and new concepts like actor, goal, plan, resource, dependency and belief played their role on operational basis. The concept of architectural style, found its place in agent oriented system gradually, and different styles evolved in agent oriented domains were presented. In general the presented styles in agent oriented environments could be categorized in four groups:

The first, are the classic styles\(^7\) which consider only one agent and assess its activity in an isolated manner. The first style was layered which lacked efficiency in implementation of an agent; the second was task tree style\(^8\) that dealt with the decision making manner and the last one was the control loop style initially presented for simple reactive agents.

The second is based on the shared memory founded on black board style with a focus on how agents communicate with one-other by applying shared data. This group consists of event based blackboard\(^9\), reflexive blackboard\(^10\), Reactive and Coordination styles\(^11\).

The third is based on organizational theory\(^12\). Organizational theory is a discipline which deals with the structures and designing identities like social identities. This theory presents a pattern to formalize structures and behaviors. In this domain the styles like Structure-in-5, Joint venture, Bidding, Pyramid and Arm’s-Length are known. And final group consists of 6 architectural styles: such as Synchronous Centralized, Synchronous Hierarchical, Synchronous Distributed, Asynchronous Centralized, Asynchronous Hierarchical, Asynchronous Distributed, with respect to distribution and the manner of communication\(^13\).

3. Artitecheral Style and its Sub Sections

An architectural style is a collection of architectural design decisions applicable in a given development context, assigned to a specific system in a particular aspect which extracts the advantageous features of every system\(^4\).

The essence of difference here is what separates style from pattern. In the context of style, the details are not discussed and a particular structure is introduced with no pattern for designing. Using style can be named a way of using basic experience of previous designers; principles discussed in style are concerned with the sub-system working upwards to higher levels in a program, something advantageous and practical.

To define a style 4 sub-sets of style should determine: The component consists of each part of the design, for instance any object in object oriented style are components. The connector- anything that bounds the parts and facilitates communication. Connectors can perform as communicators, coordinators, converters and facilitators. In object oriented style the connectors are the messages interchanged among the objects. The set of configuration rules consist of the topological ordinances which determine legitimate compositions of elements; for example, it is possible to make a rule where each element can connect to only two other elements and or the connection is one way. The semantic interpretation indicating that the compositions of designing elements framed in perceptible manner and allow for the structured systems to be analyzed formally or informally.

3. 1 Agent Oriented Concepts

It is necessary to conceptualize the primary concepts applied in agent oriented systems\(^5\).

Actor is an entity or a strategic goal. An actor can be a physical agent or even be exposed as a role or a place.
The physical agents are tangible and concrete. In the real world they are complicated physical agents like a robot or a fire extinguishing sensor. Software agents due to their tangibility have significant affect in the software engineering world in comparison with real world physical agents. The reason is obvious, since it is rather impossible or very difficult to implement physical agent with hardware. Interactive agent, record sales agent and also all of the characters in a video game are software agent. The software or physical agent is equipped with tools like sensor or operator, including features like autonomy, communicative reactivity ability, and deliberation.

Environment is what surrounds an agent and the agent can realize it. For example, for a robot the real world is its environment likewise, for a software agent other software components are counted as environment.

Sensor is a tool which makes the environmental changes understandable for the agent. No agent can make a decision without realizing its environment. The conception of a sensor for a physical agent is tangible, for example the sensor of fire extinguisher could be a thermometer or smoke detector and for a robot it can be a camera or microphone.

Operator is what the agent can change its environment by. If we imagine a personal computer as an agent, all of its inputs will be considered as sensors and all of its outputs as operators. In the example of fire extinguisher, the out controller is water and for a robot, its hands, feet, talking devices.

Self autonomy refers to the act of an agent’s operation automatically by itself despite the fact when one agent communicate with other agent even if there is a central decider between agents with respect to the goals and the reactions thereof. The level of autonomies is different, some of the agents are allowed to deny while some are not. Self autonomy of an agent does not only depend on its behavior but on its available resource and its affiliated organization.

Goal is divided into two parts, soft and hard. The hard goal is what is referred to with its functional properties. For example, hard goals of the sale system consist of sale recording, receiving reports and practical computations. The soft goals refer to nonfunctional features of the system, that is its marginal role in planning, and if ignored the project will fail and they are: speed, reliability, safety and efficiency.

Plan is illustrated at its abstract level and its execution can assure a hard or a soft goal.

Dependency refers to a situation when two actions, for any reason depend on each other. These reasons can be achieving a goal, executing some plans and releasing a resource. It should be noted that, if the goals are not fulfilled by an actor the Dependee actor cannot fulfill its own goals. There exists a dependency relation of three parts: Depender is an actor which others depend on, Dependee is an actor which depends on the Depender and Dependum is the intermediate causation between the two or more actors.

Belief indicates the knowledge of one actor on the world surrounding.

3.2 Graph Transformation Systems

One of the approaches in formal verification is the graph transformation systems (a mathematical approach that guaranty the verification of the presented model automatically). Here by adopting the graph theory and the rule based methods, creation of state space for all possible states accompanied with the ability of searching the state space the occurrence or non-occurrence of an event can be guaranteed. A graph transformation rule is made of two graphs: the LHS (The Left Hand Side) of graph and the RHS (The Right Hand Side) of graph. In fact we need to introduce two graphs to create a graph transformation rule. These two graphs usually should have similar and dissimilar parts. There are cases when both the sides are similar; while if two graphs of one rule are similar, usually they are not applied. The manner of implementing this rule is: if a state in the host graph corresponds to that of the LHS of a rule, the host graph can be converted to the RHS. The host graph: the initial model state is illustrated in a general graph form. If in one instant several rules are ready to be implemented and there is no priority defined between them, each of the rules can be implemented. In order to create a state space all rule implementation conditions must be considered. In fact this proposal model is defined by creating a host graph after which all graph transformation rules are defined. Eventually, by using the designated tools the state space is created and checked to find out whether what is expected will occur or not.

For better grasp of this concept a scenario is being presented: there is a city with some one-way roads and some passenger living in different cities. Each passenger can go to the next city if there is a route. Some of the cities are deadlocks and passengers cannot pass through them. If two passengers meet each other in one city, one of them
will be removed. To express this issue, first the host graph will be defined in Figure 1.

The above graph is made of two types of nodes where, C represent the city and P represent the Passengers, in addition to two types of edges: the first one is R meaning the road between cities and the second one is At showing where each passenger is from. After defining the host graph the rules are made. First, the motion rule will be defined for node P. In order to illustrate the rules, the approach presented is applied. The sign × indicates the condition where no LHS should exist in the graph.

- **First Rule:** The motion of a passenger from one city to the next
  This rule is charted in Figure 2.
- **Second Rule:** If two passengers meet each other in a city one of them will be removed (Figure 3).

Although these rules look simple, they can describe graph transformation systems. Generally graph transformation rules can be categorized in 4 groups: the first group contains some parts present in both the LHS and RHS. These parts stay constant in host graph as the rule is implemented. The second group contains some parts shown by × in LHS which, meaning that for the rule implementation these parts must not exist. The third group contains some parts available in LHS but not in RHS: if the rules are implemented these parts are removed from the host graph. And the last group contains some parts available in RHS but not in LHS. If the rules are implemented these parts are added to the host graph.

### 3.3 Graph Transformation Tools

Different tools are presented with this respect to their use in particular field such as AGG and GROOVE. The GROOVE tool will be used in this modeling. The most important ability of GROOVE is forming the state space tree with the ability to check the CTL and LTL and guarantee the occurrence or non-occurrence of an event. An interesting point that can be mention about GROOVE is that in this software the LHS and RHS are shown in the same graph; while in the Graph transformation theory they are shown in two graphs.

As a symbolic rule, see Figure 4.

Each one of these 4 objects A1, A2, A3 and A4 has different meanings. A1 is the same in both parts that is, it is a necessity in rule implementation with no changes imposed to it. A2 is a required factor in implementing the rule. A3 is essential for rule implementation after which it is removed from the host graph. A4 is added to the host graph after implementation of the rule. The symbolic rules applied in GROOVE are illustrated in Figure 5.

### 4. Proposal Style

In the related literature the following 4 styles Control Loop, Layered, Blackboard and Structure-in-5 are in this study. The attempt is made here to combine concepts evolved from the above mentioned styles with respect to environment with an abstract perspective and the dynamism in agent decision making manner and propose a
new architectural style in Graph Transformation Tools according to Figure 6.

This architectural style consists of 8 components. The first one is Sensor. This part connects directly to its environment and makes the responder aware of any sensed changes. In fact there is a dependency relation between responder and sender with the Dependem as sensing. Based on the type of the agent the sensor can be very simple like a smoke detecting sensor or a microphone or very complicated like an electronic eye with image processing ability. The notable point here is that the belief that an agent has in the environment is found on the quality of the sensor. Next component is Actuator the duty of which is performing defined activities. This unit is under responder control exclusively and based on the agent style, it can be very simple like sprinkler or very complicated such as hands and foots of a robot. It must be noted that actuator’s function is to change environment and this change can be sensed by sensor immediately. The next component is called Responder and its duty is to create new task. Here, task means an order that has to be observed by the Actuator. Responder can create a task directly without any permission from higher components only when the sensor of the agent is of the simple type. This component under certain conditions can be dependent or independent: if independent, it can decide on its own. The next component is the Analyst which is a logical decision making unit acting based on the received data from the sensor as follows: if it evaluates the environment as a deterministic aspect (deterministic environment is the one where the reaction of the agent against it is not related to the previous attempts) it allows the responder to react and in due course if its analysis on the environment is sequential, it will save the information in its database knowledge and hands the duty of deciding to supervisor. For implementation of this style in different scenarios many different decision making agents can be considered for the analyst. The next component is Resource Manager with a duty to manage the resources. The analyst controls this component and the created duty by responder must receive its needed resources to be implemented by the activator. The next component is the Supervisor which plays the role of a directing manager and it is responsible for several duties. Its first duty is to decide for the information that is sent by the analyst. If the information is about the internal plan of the agent, the analyst will allow
responder to create task, that is, the potential occurrence is not related to other agents but if the received information is about the general goal of other agents, in addition to above allowance it asks the sender to send information to other agents. This component is responsible for managing the receiver as well. The next component is the Sender and its duty is to send information to other agents under supervisor control and it keeps doing its job until it become sure that all agents have received the sent information, and finally the last component here is the Receiver with the duty of receiving information from other agents and by sending data to supervisor and subsequently to the Analyst, it facilitates the additional received information to the agent strategic knowledge which leads to a change in the agent's belief.

A combination of 19 graph transformation rules that are extracted from this experiment. In due course some of them will be noted (Figure 7 and Figure 8) and the rest will be presented in http://www.drivehq.com/file/DFPublishFile.aspx/FileID1739688289/Keyne04r1qasuyn/Style-Tools-Usermanual.rar

- **Rule:** if there is no dependency relation between responder and analyst (this state occurs when the agent is of the simple type and recognizing this matter the supervisor's task) the responder directly create task without any permission from higher components. This created task will be accompanied with the supervisor ID. This is when the created task, before implementation depends on the resource manager for source allocation and afterward this dependency relation is removed.

- **Rule:** if the needed resource for the task is available, the designated source will be allocated to the task by the resource manager. This phenomenon leads to 3 changes: first, the resource is not free anymore, second, it is not under resource manager control and third the dependency relation between task and resource manager will be removed.

4.1 A Case Study: Fireman Robot

This proposed style would be actualized when an example (not necessarily limited to a specific one) in the real world example. Several models can be simulated in agent oriented domain by this proposal style such as hardware and software agents. Regarding the hardware agents too much attention has been devoted to the audience which is in more conformity with this proposed style. For this purpose the fireman agent is selected. An in depth interview with an expert fire chef is made and flexible parameters discovered in order to confirm this model. Moreover, many existing standard manuals are checked by the authors here in this respect. The Scenario: A few fireman robots are communicating to one another. A fire is noted by one or more agents. For instance, two kinds of inflammable material in liquid and solid forms are considered here. The liquid form needs water and the solid, foam to be extinguished. In this scenario there are two types of sensors, the smoke, a simple sensor and advanced sensor. The agents detect 3 degrees of fire: low, medium and high. The agents collaborate with one another to extinguish the fire (Figure 9).

Fireman agent is in operational environment and its special sensors are able to recognize the fire, while there are some agents with different kinds of Fire sensor in the environment, for example a sensor of an agent may only recognize the smoke or the blaze through changes in the temperature or a complete robot can sense changes with electronic eye and image processing. At any rate

Figure 7. Task creation rule.

Figure 8. Resource Allocation Rule.
the received information by the sensor is sent to Fire Controller which creates a task for extinguishing the fire. The information can be sent from fire controller to the Firefighting Analyst if necessary. If the material in the fire is solid and there is some water in the resource, it is allocated to the Extinguisher and if the material in the fire is liquid and there is some foam in the resource, it is allocated to the Extinguisher. This is called the Extinguisher Material Manager duty. The extinguisher will extinguish the fire if the resource is allocated to the task. The firefighting analyst will allow the controller to make tasks if it senses the expansion of fire is limited. The allocated resource will be released after the task of extinguishing is completed the fighting analyst analyzes the expansion of fire and if the data indicate the size of the fire is beyond control it will add the information to its database and transfer it to the Fire Man Agent to be decided upon. After receiving the information from firefighting analyst the fire man agent let the fire controller make a task. After analyzing the data if the fire man agent senses that this task is beyond its abilities, the duty of making other agents aware will be upon the Alarm Box through which one agent can contact other agents. Other agents receive the alarm by their Alarm Detector and then create an internal help alarm by their fire man agent component. The help request made by fire man agent is transferred to firefighting analyst where the request is registered in the analyst’s database knowledge. If the agent sensor is turned in to smoke sensor the dependency relation between fire control and firefighting analyst will be removed and the fire controller can become independent in performing its tasks. If the simple agent sensor is turned in to an advanced agent sensor, the above mentioned relation will be created once again.

To implement the function of the fire man robot all the basic rules are determined based on presented style. In due course some of them will be noted (Figure 10 and Figure 11) and the rest will be presented in http://www.drivehq.com/file/DFPublishFile.aspx/FileID1739688289/Keyne04r1qasuyn/Style-Tools-Usermanual.rar

- **Rule:** Creation of a task for Fire Extinguishing
- **Rule:** if the material burning is liquid and there is foam in the extinguishing material, this will allocated to the related task. The essence of this rule is that it can be defined for different inflammable materials and or it can create a parametric rule that could cover the rules.

![Figure 9. Metadata of the Fireman Robot.](image)

![Figure 10. Task Creation for Extinguishing Rule.](image)

![Figure 11. Resource Allocation Rule (Foam).](image)
5. Model Checking

What distinguishes the formal and informal methods is their function pattern based on their mathematical approaches, that is, if one feature is checked in formal method, the obtained result will be 100% reliable in the manual Approaches the verification may not provide us with a final answer. What is implemented here with respect to graph transformation system is the formal architecture style software. No checking is conducted on the model and its function accuracy is not assessed.

According to the GROOVE functionality which is based on creating state space the strategy here is to analyze and renew the whole space state and since all possible states are assessed there will not be error. At this point prior to anything some characteristics are assessed which can be verified in different systems. These characteristics are: Safety which means a bad event would never happen (possibility zero), Livens, which means a good event would happen in any way (possibility one), Fairness, which means a good event happens continuously in all states and Reachability, which means that it is possible for a good event to happen.

To describe these features, a scenario named war is simulated here. In the war death incident must not happen under no circumstances where all safety features must be considered. Victory, the objective of a war. It should be achieved anyhow, but it does not mean that the victory status is enjoyed every moment but they since it must be achieved Livens is probable. The feature of “carry a gun at all the times” for a soldier for his safety is a must and this is called fairness. Becoming commander is the possibility the might happen or not with a soldier, while being aware of both the possibilities affects his emotions, hence the reach ability.

In GROOVE tool there are two temporal logics: the CTL and LTL in order to search the state space. We prefer to choose CTL because it is more simple and linear than LTL\(^3\). At first pay attention to the definitions presented for CLT. E represents some of the paths in the state space tree, A represents all the paths in the state space tree, F represents some of the states in the path or in designated paths in the state space and finally G represents all states’ space in path or the designated paths. By combining these concepts and graph transformation rule will obtain following formula. EF (rule1) where at least in one path and at least in one of the states rule1 is implemented. By using this expression the livens can be assessed. This is the most applicable expression. EG (rule1): implementable at least in one path at all states of Rule 1. None of the mentioned features can be modeled through this expression and its applicability is low. AG (!rule1) we can prove safety. AG (rule1): rule1 is implemented in all paths at all states. By using this expression fairness can be assessed and by altering it to AG (!rule1) safety can be assessed as well.

To verify this model by considering the state space created by GROOVE tool, the CLT temporal logic is used. For this purpose different issues are addressed; in order to them some particular rule must be introduced to verify the model. Said otherwise, these rules do not add or reduce, but just verify. Both LHS and RHS remain the same and according to graph transformation theory they are not rules. Two of the addressed issues are; in first issue an occurrence is expected; hence, a rule must be written and implemented on all paths at least once based on the temporal logic, through expression AF(rule1) (rule1 is the written rule for each question). Now the question If the extended fire detected by an expert agent, would the neighboring agent become aware? (Figure 12).

This question evaluates a state which should not occur even once in space state; therefore, expression AG (!rule-Name) is applied here. Now the question: Is it possible for a task created by an agent, to be implemented by another agent by a mistake? (Figure 13).

![Figure 12. Proofing Farness Rule.](image-url)
6. Conclusion

Each one of the existing styles in agent oriented environment analyzes the concept of style in a specific view point with no concern on other aspects. Thus it is necessary to design a style which would consider all aspects of an agent oriented system. In this proposed style more details are discussed on and by observing the basic concepts of style, this style is closer to the architectural pattern. In this study all the aspect of previous presented styles in agent oriented systems are analyzed. This style is modeled through the graph transformation systems using the GROOVE tool. Different kind of modeling could be presented based on this proposed style, but the fireman robot is selected. Although using the graph transformation system provides a practical viewpoint for the user and reserve the abstract nature of the issue, it is not significant to use graph transformation systems without using the ability of formal automatic verification assessment model, thus the modeled example is finally verified by GROOVE tool.

The achievement of this article is the importance in use of graph transformation system in expressing an architectural style. There are some advantages and disadvantages in modeling style trough graph transformation systems. The disadvantage here is its being time consuming. The advantage is not only the ability to verify the model automatically, but sparking new ideas in the researchers mind. The attempt to conduct this study and introduce this proposal was initiated in the authors’ mind through the idea of combining the 4 available styles.

If only the aspect of expense and verification are of concern it will not be necessary to model several systems using graph transformation systems, because this method is costly and can be used only when there is high risk on human life and high damages. Usually in normal systems adopting formal methods are not economically feasible but if exports spend more time on studying this mater, a gradual decrease in costs of formal methods will be guaranteed and the formal method could become applicable in all systems.

7. Reference


