IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received July 18, 2020, accepted August 2, 2020, date of publication August 6, 2020, date of current version August 19, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3014694

An Organizational Structure and Self-Adaptive Mechanism for Holonic Multi-Agent Systems

MEIJIA WANG, QINGSHAN LI, AND YISHUAI LIN
School of Computer Science and Technology, Xidian University, Xi'an 710071, China

Corresponding author: Qingshan Li (qshli@mail.xidian.edu.cn)

This work was supported in part by the National Natural Science Foundation of China under Grant 61972300, Grant 61902288, Grant 61672401, and Grant 61373045; and in part by the Pre-Research Project of the Thirteenth Five-Year-Plan of China under Grant 315***10101 and Grant 315**0102.

ABSTRACT A holonic multi-agent system combines the concept of a holon with a multi-agent system; this combination has been proven to be an effective way to build a complex system. Great progress has been made in this area, but previous studies are fragmented and lack of a task-based perspective to model different systems in the real world. Therefore, this article proposes a formalistic model for HMAS from a task-based perspective. Not only the static organizational structure is designed, but also the dynamic running mechanism, including the self-adaptive mechanism and the task assignment mechanism based on the proposed holonic structure, are also discussed. Finally, a case study is provided to verify the self-adaptive mechanism. The experimental results show that our proposed DHMAS has the ability to adapt to the changing environment, and performs better in terms of the success rate and the response time when the system is heavily loaded.

INDEX TERMS Holonic multi-agent system, multi-agent system, organizational structure, self-adaptive mechanism, task-based perspective.

I. INTRODUCTION

The open and heterogeneous network environment makes the scale of software larger and larger, and the distribution of software more and more extensive, which leads to an increase in the complexity of software systems [1], [2]. Next-generation software systems have exhibited some typical characteristics, for instance, a large number of functional components, distributed control and storage, nonlinear processes, dynamic and open operating environments, and unpredictable unit interactions [3]–[6], which highlight the needs for new approaches of software system development [7]–[11]. A multi-agent system is a computerized system composed of multiple interacting intelligent agents within an environment. A Holonic Multi-agent System (HMAS) refers to a multi-agent system in which the agent is assigned to a self-similar nesting called a holon. It has been proved that HMAS is a promising approach for complex system design, because of the characteristics of flexibility, intelligence and scalability it offers [12]–[14]. Nevertheless, due to the relatively young age of the researches in HMAS, the approaches for designing the software systems based on HMASs are still immature [15], [16]. The previous studies are fragmented and lack systematic. To achieve system goals, HMAS must perform different tasks from the environment. However, the perspective of task-based model is ignored when the existing studies discuss HMAS. Therefore, this article studies HMAS from a task-based view, the research problems we want to discuss are given below:

①. *Organizational Structure*: How to design a flexible holonic structure? How to manage a holon? How to model the required information for a holon? For instance, how to define the cost, and how to represent a task?

②. *Self-adaptive mechanism:* For complex software systems, the changes of environment and user demands that may occur during the runtime. Therefore, how can HMAS based on a specific organizational structure provide a self-adaptive mechanism to adapt to the new requirements in a constantly changing environment? When to trigger the self-adaptive mechanism, and which adjustments should HMAS make?

③. *Task assignment mechanism*: Dynamic task assignment is a problem that HMAS must solve. The change of environments outside the system requires the system to allocate new tasks that occur dynamically, and the changes in the

The associate editor coordinating the review of this manuscript and approving it for publication was Porfirio Tramontana.

computing resources of members within the system will also affect the allocation of new tasks. Reasonable task assignment mechanism is significant for HMAS, helping save system cost and improving resource utilization. However, there is less study on the task assignment mechanism for HMAS. A task environment has a direct impact on the management of the organization. Therefore, for a specific organizational structure, how to reasonably assign new tasks to improve the resource utilization rate of the system?

This article makes an attempt to explore the above issues by proposing a formalistic model for building complex and dynamic software systems based on HMAS. Our contribution is that, from a task-based perspective, we discuss the static organizational model and dynamic running mechanism of HMAS. The static organizational structure is designed. The dynamic self-adaptive mechanism which ensures the successful execution of the tasks, and the task assignment mechanism based on the proposed organizational structure, are also presented. The HMAS based on our proposed static organizational structure and dynamic running mechanism is called Dynamic Holonic Multi-agent System (DHMAS). We also provide a case study, that is, a metasearch engine based on DHMAS. The experimental results show that DHMAS has the ability to trigger the self-adaptive mechanism to adapt to the new requirements. Compared to the baseline, the proposed DHMAS guarantees the success rate and the response time of the system.

The rest of this article is organized as follows. Section 2 introduces the related work. Section 3 describes the details of DHMAS, including the organizational structure, self-adaptive mechanism and task assignment mechanism. Based on the proposed DHMAS, Section 4 provides a case study and the experimental results. Finally, conclusions and future work are presented in Section 5.

II. RELATED WORK

Holonic Multi-Agent Systems evolved from Multi-agent Systems which based on hierarchy structure [17]. HMAS has attracted the attention of scholars in many fields, such as intelligent transportation systems [18], distributed sensor network management [19], supply chain management [20], urban traffic control [21], *et al.* A lot of researches focus on how to use HMAS to solve problems in a specific application. Reference [22] has analyzed the holonic paradigm in biological network simulation to employ their abilities, such as self-organization, autonomy, distribution and so on. An internal holon architecture is designed. Each holon has all of the agent abilities in its agent part. Moreover, it has a rule engine and a knowledge base. The decision center of the holon makes use of its rule engine to decide in different situations. This knowledge base helps the holon to distinguish between its upper- and lower-level holons. [21] has used a holonic multi-agent system to model a large traffic network to reduce the complexity of the system. A traffic network containing fifty intersections is partitioned into a number of regions, and holons are assigned to control each region. The holons are hierarchically arranged in two levels, intersection controller holons in the first level and region controller holons in the second level. [23] has focused on the notion of the role of a Holonic MAS (HoloMAS) and its contribution to the adaptive control of manufacturing systems. A holonic multi-agent system (HoloMAS) using roles to provide an adaptive control system for manufacturing systems is proposed, and the HoloMAS proposal is also validated using simulations and through a real implementation on a flexible assembly cell. Reference [24] has presented the application of holonic organization to reduce multi-agent system complexity in the modeling of a large SIP network. Holonic organization is a hierarchical structure in which each holon covers a geographical area of the SIP network at the first level. At the second level, upper-level holons control the first-level holons, and so on. Overload control is achieved by communication and the exchange of knowledge between the intelligent holons. Experimental results show that the proposed method prevents overload in the SIP network. Reference [25] has presented a goal-based holonic multi-agent system (HMAS) for the operation of power distribution systems and discussed various operating modes and associated goals. Then, the role of HMAS is demonstrated for two applications in distribution systems, one associated with control of reactive power at solar photovoltaic installations in individual homes for optimal operation of the system and the other dealing with state estimation of a system leveraging different measurements available from smart meters in homes. Reference [20] has proposed a preliminary framework of a holonic multi-agent model for supply chain management in the oil industry, synthesized the basic concepts, and analyzed a case study. Reference [19] has utilized a holonic multi-agent system to study control architectures and control methods that are applicable to the management of sensors for tactical surveillance. The hierarchical and recursive structure of a holonic architecture provides the required flexibility and robustness without deviating significantly from the current military command structure.

Some researchers studied the self-organization mechanism for HMAS. Reference [26] has introduced a holonic framework to model and engineer complex systems., the concept of capacity is prposed as the description of agents know-how. The role specification is based on the description of required know-hows, described using capacities. To play a role, a holon has to possess an implementation (that would be specific according to its architecture) for each required capacity. In the proposed approach, the super-holon has the ability to obtain new capacities from the collaboration of its members by instantiating specific organizations. Finally, self-organization mechanisms allowing a holon to dynamically change its set of capacities and so achieve its new goals. [27] has proposed a self-organizing algorithm for holonic multi-agent systems, which is based on the local information of the agents about others agents they can communicate with. The proposed method assumes an interaction networks among the agents of a multi-agent system and constructs

the holonic structure in a bottom-up approach, which used common social concepts. There is no central unit for building and controlling the holarchy in this model, and the whole process is controlled by the member agents, according to their local information about themselves and their neighbors in a multi-agent network. Reference [28] has proposed an evolution to the ADACOR holonic control architecture inspired by biological and evolutionary theories. A two-dimensional self-organization mechanism is designed taking the behavioural and structural vectors into consideration. The behavioural self-organization, found at micro-level, which allows the system to respond smoothly to perturbations, and the structural self-organization, displayed at macro-level, which lets the system react more drastically. Reference [29] has introduced a suggestive model for holonic systems, callled Holonic Social Systems (HOSSs). A class of Petri nets is developed to control and manage physical resources and information data. To design the holons, all aspects of the system should be analyzed using the Petri nets. By using fuzzy set and uncertainty theoretical concepts, [30] has constructed a mathematical foundation for modeling MAS, where appropriate holonic structures are identified. This approach opens new possibilities for the design of any distributed system that needs self-organization as an intrinsic property.

Holonic multi-agent systems exhibit flexibility, intelligence, and hierarchy, and they are significant for the building of complex systems. However, most previous studies focus on how to use HMAS to model specific systems in various application areas. Although some researches pay attention to the self-organization mechanism for HMAS, the existing studies are still fragmented to model different systems in the real world and lack of task-based perspectives. The research contents of these related works are analyzed, as shown in TABLE 1. It can be seen, except [27], other studies have not fully discussed the organizational structure, self-adaptive mechanism, and task assignment mechanism for HMAS. Although [27] studies these three aspects in a task environment, it does not consider how to adapt to the environment when task failed. Therefore, this article proposes a formalistic model for HMAS from a task-based perspective. Our contribution is that, not only the static organizational structure is designed, but also the dynamic running mechanism is discussed, including the self-adaptive mechanism which ensures the successful execution of the tasks, and the task assignment mechanism based on the proposed holonic structure.

TABLE 1. Research contents of the previous studies.

Organizational Structure	Self-adaptive Mechanism	Task Assignment Mechanism	Task-based Perspective
[18] [19] [20] [21] [22] [23] [24] [25]	[18] [22] [23] [24] [25] [26]	[19] [27]	[27]
[26] [27] [28] [29] [30]	[27] [28] [29] [30]		

III. PROPOSED MODEL

A complex system implements various tasks to achieve the system's global goal, and the success rate of task execution has an important impact on the performance of a system. Due to changes in the external or internal environment, new tasks may continue to be generated in the system, or the tasks that have been assigned cannot be completed, etc. Therefore, it is significant to study a system from a task-based perspective. However the previous studies about HMAS seldom describe the system by a task-based model. Therefore, this article proposes a novel Holonic Multi-agent System for complex and dynamic systems in a task-based perspective. In this section, the organizational structure of DHMAS is introduced. Furthermore, the task assignment strategy and the self-adaptive adjustment mechanism are designed as well.

A. ORGANIZATIONAL STRUCTURE

The holon is the core of DHMAS. The well-designed holonic structure effectively combines the holons and MAS. In HMAS, a holon often provides a specific function. The structure of a holon consists of three parts, as shown in FIGURE 1.

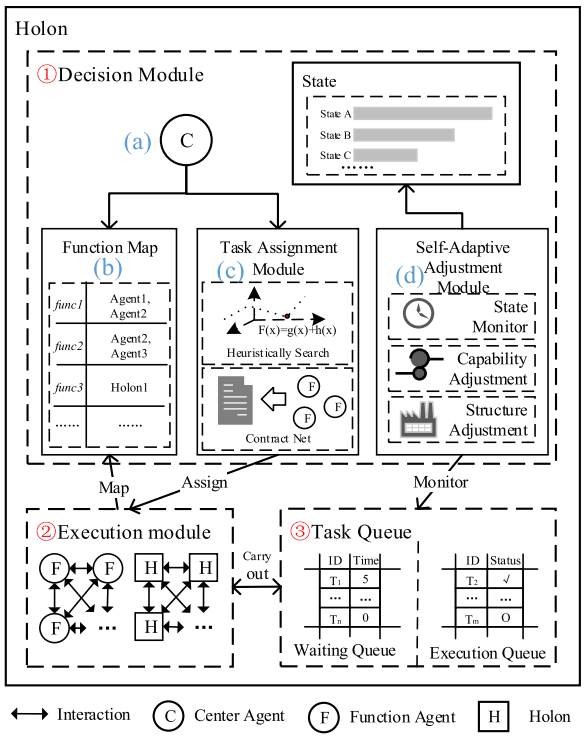

FIGURE 1. Model of a holon.

①. a Decision Module, as the intelligent control center of a holon, it is responsible for recognizing and decomposing tasks, monitoring states, adjusting the competitiveness and structure adaptively, and assigning the tasks to execution units (holons or agents) according to the function and competitiveness of each execution unit in the Execution Module. The Decision Module has the global information of a

holon, to ensure the successful execution of task. Moreover, the Decision Module includes: a). a Center Agent, receives the environmental requirements, and decomposes the task into several subtasks. The Center Agent also plays the role of the head of the holon, providing the means for communication with the external holons and function agents. The Center Agent is responsible for assigning tasks to the execution units according to the task assignment strategy generated by the Task Assignment Module as well; b). a Function Map, which is the functional yellow page, recording the function of the current holon and providing the ability to communicate with the execution units that perform the subfunctional task. Moreover, the Decision Module monitors the change of the competitiveness of each execution unit based on the record of the Function Map. When the structure of the holon changes, the Function Map must be modified at the first time. Therefore, the Function Map corresponds to the structure of the holon in real-time. Based on such constraints, the Task Assignment Module can find an execution unit efficiently; c). a Task Assignment Module, which is the task assignment center of the holon. A task is composed of a set of subtasks with a specific collaboration order at first. Then, according to the task assignment algorithm, the Task Assignment Module generates a task assignment strategy to find the appropriate execution units to perform the subtasks; d). a Self-adaptive Adjustment Module, which is the center for monitoring and adjusting the state of the holon. It monitors the Waiting Queue and Execution Queue in the task queue, dynamically adjusts the competitiveness of the holon in real-time during the running of the system, and feeds back the change of self-states to its parent holon through the Center Agent. Meanwhile, when some tasks failed to execute or the waiting tasks have been polarized, the Self-adaptive Adjustment Module will trigger the structure adjustment of the holon to achieve the purpose of adapting to the environment of the current task by scheduling the computing resources from other execution units.

②. an Execution Module, is composed of several indivisible Function Agents or Holons at low levels. It is responsible for completing a specific functional task. As shown in FIGURE 1, to complete a task, these Agents and Holons might interact with each other by sending messages. Each Function Agent still retains its own perception, and autonomy. When the assigned tasks conflict with the current local environment, the Function Agent can refuse to perform the tasks according to the actual situation and return the results of task failed to the parent Holon. At this time, the parent Holon will adjust its structure to ensure the task could be executed successfully.

③. a Task Queue, consists of a Waiting Queue and an Execution Queue, reflecting the current load state and task execution progress, respectively. A holon obtains changes of the competitiveness according to the status of the Task Queue.

The organizational structure of DHAMS is shown in FIGURE 2, which is an example of a three-level hierarchical structure. Holons in level 2 receive the system requirements from the environment. The Decision Module which includes

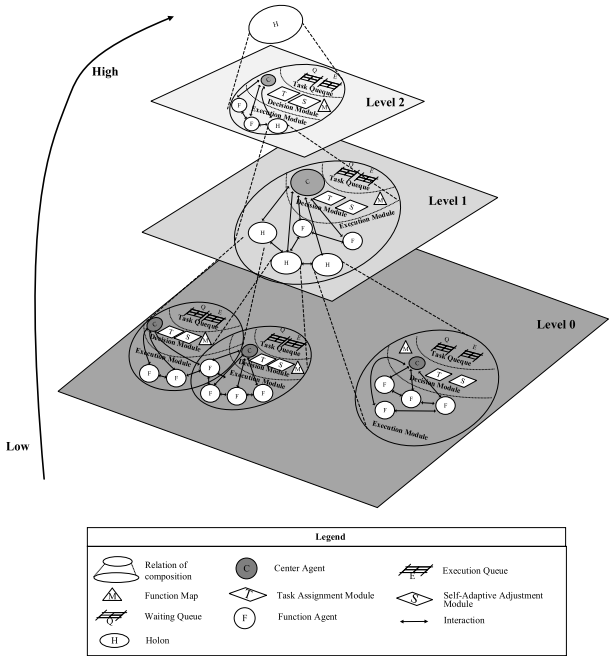

FIGURE 2. An example of a three-level hierarchical structure of DHMAS.

a Center Agent, a Task Assignment Module, a Function Map, and a Self-Adaptive Adjustment Module, is responsible for decomposing the tasks and assigning the subtasks to the Function Agents and the sub-holons which in the Execution Module. The sub-holon in level 2 also includes a Decision Module, an Execution Module, and a Task Queue. When the subholon receives the tasks, the Decision Modules in the subholon will determine whether the tasks need to be decomposed further, and assign the subtasks to the Function Agents and the subholons in level 1. After receiving the tasks, the subholons in level 1 will first analyze the tasks, and their Decision Modules will decompose the tasks, and assign the tasks to the Function Agents in Level 0. Because the Decisions Module of the subholons found that all the tasks could be performed by the Function Agents in Level 0. The tasks which received by Level 0 will no longer decomposed, and a three-level hierarchical structure is built. It is worth noting that a Function Agent and a subholon could belong to different parent holons, as shown in Level 0 in FIGURE 2.

1) THE DEFINITION OF FUNCTION AND COMPETITIVENESS

Function and competitiveness are the important attributes for a Holon.

Each execution unit provides a function based on its skill. The description of a function at least contains a name. For example, a Holon has the skill of analyzing a user's interested webpages, then the function of this Holon can be named as *UserInterestsObtaining*.

The function provided by a holon in DHMAS is uniquely determined by the structure, and a function may be composed of some subfunctions from subholons or Function

Agents. For example, holon H_0 provides function X, written as *Func_X*. Moreover, function X is composed of function Y provided by holon H_1 and function Z provided by holon H_2. Therefore, the function of H_0 can be expressed as (1):

$$Func\_X = \{Func\_Y, Func\_Z\} \tag{1}$$

The functions in DHMAS can be divided into two categories. One is a composite function, represented as "*Func*"; such functions are composite functions that can be further divided. The functions of most holons are composite functions. The other type is an atomic function, represented as "*func*"; such functions cannot be further divided. In DHMAS, only a Function Agent performs the indivisible function task.

An important attribute of the function is "cost." The cost describes the resources or time that a holon needs to consume when completing a function. When the available resources of the holon are greater than the cost of a function, the holon can schedule its resources to perform the function. In contrast, if the available resources do not meet the requirements, the holon will not execute the function at first but will execute it after resources are released by other holons. For atomic functions, the resources or time that must be consumed are determined by a specific system. For composite functions, the cost is the sum of the costs of all subfunctions.

The competitiveness describes the computing resources owned by an execution unit. For example, a Holon *H0*, its competitiveness of function *Func_X* is $5r$, where r refers to the average resource per unit. It means when *H0* implements function *Func_X*, the maximum number of resources that can be consumed is $5r$.

The more available resources the execution unit has, the stronger its competitiveness is. In general, the competitiveness is directly related to the structure and function it provides. For a Function Agent, the function it provides is indivisible. Therefore, its competitiveness is equal to the cost of executing a specific function task. The number of functions that an Agent or a Holon can implement is related to its competitiveness. Execution units complete specific tasks to implement specific functions, and each task has its own cost. An execution unit can only implement the functions that the required costs are less than its competitiveness. The larger the competitiveness of an execution unit, the more complex functions can be complemented.

A composite function is a complex function provided by multiple subholons or function agents through a specific collaboration. Therefore, when evaluating the competitiveness of a holon, it is necessary to consider its subfunctions. For example, the function of a holon, written as *Func_X*, is composed of subfunctions *Func_Y* and *Func_Z*. Inside, the number of subholons that can currently provide function *Func_Y* is n, and the number of subholons that can currently provide function *Func_Z* is m. Then, the competitiveness of function *Func_X* provided by the holon is calculated by (2).

$$\begin{aligned} &Competitiveness\_Holon^{X} \\ &\quad = \sum_{i=1}^{n} Competitiveness\_Holon_i^{Y} \\ &\qquad + \sum_{j=1}^{m} Competitiveness\_Holon_i^{Z} \end{aligned} \tag{2}$$

Meanwhile, the competitiveness of the holon regarding function Holon^{Y}, represented as $Competitiveness\_\text{Holon}^{y \in X}$, is calculated by (3).

$$\begin{aligned} &Competitiveness\_Holon^{y \in X} \\ &\quad = \sum_{i=1}^{n} Competitiveness\_Holon_i^{Y} \end{aligned} \tag{3}$$

The function and competitiveness determine the state of a holon from qualitative and quantitative perspectives, respectively. HMAS assigns tasks based on the changes of the states of holons when a system is running.

2) REPRESENTATION OF THE TASK FOR A HOLON

A task is an abstract entity that the system needs to perform, as defined by (4).

$$\begin{aligned} Task ::= &< Task\_ID, Req\_Func, Expect\_time, \\ &State, Next\_Task > \end{aligned} \tag{4}$$

where *Task_ID* is a unique identifier for a task. The system utilizes T*ask_ID* to determine the affiliation before and after the task is decomposed. For example, the *Task_ID* of a task is "*1*", and after decomposition, three subtasks can be obtained. The *Task_IDs* of the subtasks are "*1.1*", "*1.2*", and "*1.3*". Moreover, subtask "*1.1*" can be further decomposed into two subtasks, whose *Task_IDs* are "*1.1.1*" and "*1.1.2*", respectively. In this manner, all tasks will be numbered. When the task is completed, the task will be synthesized according to the numbering sequence.

Req_Func refers to the functions that a holon needs to implement to perform the task. These functions correspond to the functions of other holons, that is, each task can find one or more corresponding holons to ensure that the task executes smoothly. At the same time, the execution of each task comes at a cost, which is the sum of all costs of the required functions.

Expect_time is the expected time required to complete a task, that is, the maximum time spent on the task. The *Expect_time* of a task is related to the type of a task and user requirements, and it specifies the deadline to finish the task. For complex tasks that need to be decomposed into subtasks, the expected time of the subtasks are calculated by (5), where *Waiting_time* indicates the waiting time of the task. The holon records the time of accepting the complex task, written as *Accept_time*, and the task is decomposed and assigned at *Current_time*. Then the *Waiting_time* can be defined as the difference between the *Current _time* and *Accept_time*. If there is a dependency between some subtasks, the *Expect_time* of theses subtasks will be also affected. For example, a task T can be composed of *T1->T2*, and *T2* is waiting for *T1*. If the *Expect_time* of T is 10s, and the

Expect_time of *T1* is 2s, then the *Expect_time* of *T2* cannot exceed 8 seconds.

$$\begin{aligned} &Expect\_time_{sub} \\ &\quad = Expect\_time - Waiting\_time \\ &\quad = Expect\_time - (Current\_time - Accept\_time) \end{aligned} \tag{5}$$

State is the current state of the task, which is divided into three types, Completed, *Waiting*, and *In Progress*, indicating that the current task is completed, waiting, and executing, respectively.

Next_Task is a variable indicating the relationship between different tasks. If there is a sequence between some tasks, for example, the output of task A is the input of task B, the variable of *Next_Task* for task A will be filled with the *Task_ID* of task B. If the task is the last step of a task sequence or an independent task, *Next_Task* should be filled with ∅.

3) HOLONFICATION

The process of holonfication is based on system requirements. A system processes the input data from the environment and outputs the related data to meet the environmental requirements. Therefore, by analyzing the data flow of a system, the holonic structure will be easy to construct. Agents are the smallest units of a system. Each agent completes a specific function to meet a specific requirement. Agents cooperate with each other to process the data, to achieve some higher-level functions. Therefore, before holonfication, the data flow between different agents should be discussed. In our proposed model, if the data flow is processed by an agent, and flows to multiple other agents or Holons, then the agent will be defined as a diving line. It is considered that the diving line provides input data flow for other agents or Holons, then the diving line and the agents or Holons who provide data flows to the diving line will form a new Holon. The process of the holonfication includes three steps:

1. Starting from the input data of the system, according to the data flow, find the first dividing line, and form the relevant Holons, then update the data flow diagram;

2. Find all the dividing lines, form the related Holons, and update the data flow graphs;

3. According to the updated data flow graph, find the end agent, that is, the agent at the end of the data flow. Then, the end agent and the other agents or Holons in each data flow form the related Holon, respectively. Finally these related Holon forms the Holon in the top level of the system.

For example, to meet the requirements from environments, the system *S* must contain 6 types of roles. Each role must be played by an *Agent* at least, and these agents are written as *Agent1*, *Agent2*, *Agent3*, *Agent4*, *Agent5* and *Agent6*, respectively. The data flow between these 6 agents is shown in FIGURE 3(a). *Agent2* and *Agent3* are the diving lines. The process of updating the data flow graph is shown in FIGURE 3(b) and FIGURE 3(c). We can see, *Agent1* and *Agent2* form Holon *H1* when the data flow graph is updated for the first time. *H1* and *Agent3* form *H2* when the data flow graph is updated for the second time. FIGURE 3(c) is the final data flow graph, it can be seen that there are three input stream for the end agent *Agent6*, *H1->Agent5->Agent6*, *H2->Agent6*, and *H2->Agent4->Agent6*. Therefore, *H1*, *Agent5*, and *Agent6* form a new Holon *H3*. *H2* and *Agent6* form a new Holon *H4*, and *H2*, *Agent4*, and *Agent6* form a new Holon *H5*. At last, *H4*, *H5* and *H6* form the top-level Holon of system *S*. The organizational structure of *S* is shown in FIGURE 4.

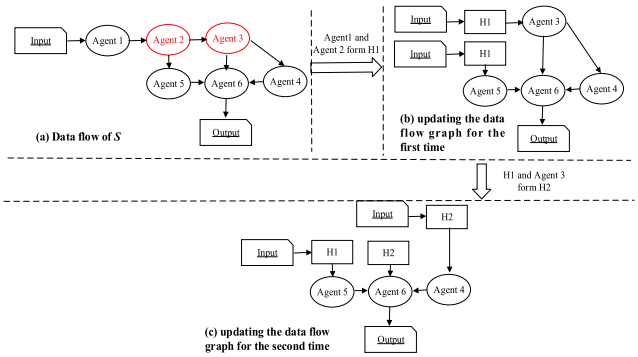

FIGURE 3. Process of updating data flow graph.

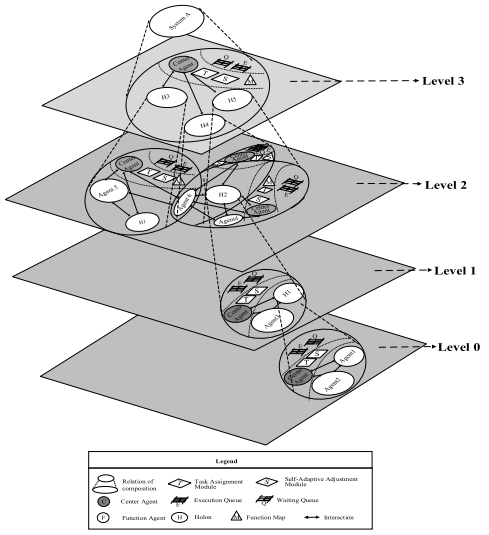

FIGURE 4. Organizational structure of system S.

B. SELF-ADAPTIVE MECHANISM

The Self-adaptive Mechanism gives the system the ability to autonomously modify its behavior at run-time in response to changes in its environment. The Self-adaptive Adjustment Module is designed to implement the self-adaptive adjustment mechanism. First, the module monitors the Task Queue to obtain the current working state of the holon. Then, according to the current working state, the module adjusts the competitiveness of the holon or even changes the structure of the execution module when necessary, and it returns the result to its parent holon through the Center Agent.

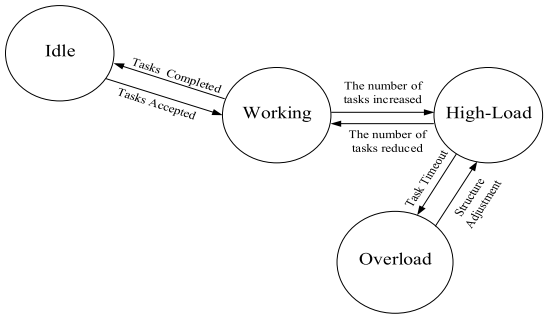

FIGURE 5. State Transition Diagram for a holon.

1) STATE MONITOR

Each holon switches back and forth between four different states starting from the system: idle, working, high load, and overload. The State Transition Diagram is shown in FIGURE 5.

- Idle: The holon has not accepted any tasks, or the accepted tasks have been completed.
- Working: The holon is performing tasks normally, and the required competitiveness of the waiting tasks account for less than 80% of the total competitiveness of the holon;
- High Load: The holon is performing tasks, and the required competitiveness of the waiting tasks account for more than 80% of the total competitiveness of the holon;
- Overload: The accepted tasks are beyond the current competitiveness of a holon, it means that the holon is unavailable.

The Self-adaptive Adjustment Module obtains the state of the holon by observing the Task Queue. The Task Queue contains two specific queues, the Waiting Queue and the Execution Queue. As the names imply, these two queues maintain the tasks in the "Waiting" and "In Progress" states, respectively.

The structure of the Waiting Queue is shown in FIGURE 6. *Task* maintains the basic information of each task that has been accepted by a holon. *Accept_time* records the time of acceptance of the task by the holon, while *Waiting_time* indicates the waiting time of the task.

Task	Accept_time	Waiting_time (min)
<1.1, Func_X, 10, Waiting, 1.2>	16:10 8/3/2019	5
<2.2, Func_P, 17, Waiting, Ø>	16:00 8/3/2019	15
.....	.....	.....
<13.2, Func_K, 13, Waiting, 13.4>	15:57 8/3/2019	18

FIGURE 6. Structure of Waiting Queue.

The Execution Queue records the decomposition of a task and the current state of the subtasks for a holon. For example, when *Task 1.1* begins to execute, the Execution Queue is as shown in FIGURE 7.

Task	Sub_task	State	Complete_time
<1.1, Func_X, 10, In Progress, 1.2>	<1.1.1, Func_x, 3, Completed, 1.1.3>	Completed	17:02 8/3/2019
	<1.1.2, Func_y, 4, In Progress, 1.1.3>	In Progress	null
	<1.1.3, Func_z, 5, Waiting, Ø>	Waiting	null
.....			

FIGURE 7. Structure of the Execution Queue.

Task maintains the basic information of each task. *Sub_task* records all of the decomposed subtasks. Because there may be a sequence between subtasks, when the subtasks are in the Execution Queue, they may still be temporarily waiting for the completion of the predecessor subtasks. *State* is used to record the current state of each subtask, and *Complete_time* indicates the completion time of each subtask. When the last subtask is completed, the current task is considered complete, and the completion time of the subtask is also the completion time of the entire task. After the task is completed, the holon returns the execution result to its parent holon.

It can be seen that the Self-adaptive Adjustment Module judges the current working state of the holon by observing the information of the Task Queue, and it further adjusts the competitiveness or structure of the holon for different runtime environments and internal states.

2) COMPETITIVENESS ADJUSTMENT

The competitiveness adjustment refers to adjusting the competitiveness value for execution units. It is a normalized self-adaptive adjustment mechanism. The idea is mainly to monitor the Task Queue through the Self-adaptive Adjustment Module and update the competitiveness of the holon according to the Execution Queue. The competitiveness adjustment is designed to adjust the overall competitiveness as well as the subfunction competitiveness for execution units, reflecting the holon's overall load and subfunction loads.

To evaluate the current workload, it is necessary to comprehensively consider the three factors of the length of the Waiting Queue, the waiting time of each waiting task, and the required functions of each waiting task. The waiting time reflects the current load state of the function unit, that is, the longer the waiting time of the task, the higher the workload of the execution unit, and the lower the competitiveness to perform tasks. The Self-adaptive Adjustment Module will determine the urgency of the task by considering the waiting time of the task and the expected time of completing the task.

The urgency is recorded as β, calculated by (6).

$$\beta = \frac{Waiting\_time}{Expect\_time} \tag{6}$$

If the *Waiting_time* is less than or equal to the *Expect_time*, then $\beta \in [0, 1]$, indicating that the execution of the task is still within the controllable range. However, as the Waiting_time grows, when $\beta \in (1, +\infty)$, the task becomes urgent, and the *Waiting_time* is greater than the *Expect_time*, which indicates that the task is beyond the controllable range and needs to be adjusted urgently.

Because different tasks require different functions and different functions take different costs, in addition to urgency, the required functions for a task also have a direct impact on the workload state. Therefore, the workload of an execution unit for task *i* is represented as $Load^{Task_i}$, as calculated by (7):

$$\begin{aligned} Load^{Task_i} &= \beta \times Task_i.Req\_Func.Cost \\ &= \frac{Waiting\_time}{Expect\_time} \times Task_i.Req\_Func.Cost \end{aligned} \tag{7}$$

Thus, the current competitiveness of a holon is determined by (8):

$$\begin{aligned} &Competitiveness\_Holon^{X}_{current} \\ &= Competitiveness\_Holon^{X} - \sum_{i=1}^{n} Load^{Task_i} \\ &= Competitiveness\_Holon^{X} - \sum_{i=1}^{n} \left(\frac{Task_i.Waiting\_time}{Task_i.Expect\_time} \right. \\ &\quad \left. \times Task_i.Req\_Func.Cost\right) \end{aligned} \tag{8}$$

where $Competitiveness\_Holon^{X}$ is the initial competitiveness of the holon, and $Load^{Task_i}$ is the workload of the holon for waiting task *i*. The competitiveness of the holon changes with the number of tasks in the Waiting Queue and the waiting time of each task. It reflects the current workload of a holon.

The Self-adaptive Adjustment Module should also monitor the competitiveness regarding each subfunction for a holon. When a new task is added to the Execution Queue, the Self-adaptive Adjustment Module updates the current competitiveness regarding each subfunction of the Holon based on the cost of the specific task. The current competitiveness regarding a subfunction for a holon is obtained by (9):

$$\begin{aligned} &Competitiveness\_Holon^{y\in X}_{current} \\ &= Competitiveness\_Holon^{y\in X} \\ &\quad - \sum_{i=1}^{m} \frac{Task_i.Waiting\_time}{Task_i.Expect\_time} \times Task_i.Req\_Func^{y}.Cost \end{aligned} \tag{9}$$

where $Competitiveness\_Holon^{y\in X}_{current}$ is the current competitiveness regarding subfunction y of a holon. $Competitiveness\_Holon^{y\in X}$ refers to the initial competitiveness regarding subfunction y. m is the number of tasks, and $Task_i.Req\_Func^{y}.Cost$ is the cost regarding function y for $Task_i$.

The algorithm of competitiveness adjustment mechanism is shown in ALGORITHM 1. The time complexity of competitiveness adjustment is algorithm is *O(1)*.

When the competitiveness of a holon changes, its parent holon monitors the change and records its current competitiveness to update the function map. The current competitiveness of a holon is significant when its parent holon finds the appropriate execution units for new tasks.

3) STRUCTURE ADJUSTMENT

Structure Adjustment refers to the increase or decrease of the number of some specific execution units in the execution module of a holon based on the current workload and environment to meet the requirements of a task and realize rational allocation of computing resources. Compared to competitiveness adjustment, structure adjustment produces a greater impact on a system. It will not only change the competitiveness of the holon, but also adjust the structure of the execution module that is not suitable for performing the current task. Because structure adjustment will bring about obvious changes of functions of the system, overly sensitive trigger conditions will cause frequent structure adjustment for the holon, resulting in the vibration of the system and affecting the work of other modules. Therefore, the trigger condition of structure adjustment has a higher threshold.

Structure adjustment of a holon occurs in only two cases:

(1). The waiting tasks have been polarized. Under ideal conditions, the subfunctions provided by the execution module of the holon are proportional to the required functions of the current task; that is, the competitiveness allocation for the sub functions of the holon is reasonable and ensures the rational use of computing resources. However, in fact, too many tasks sometimes pile up for some execution units while other execution units are idle, indicating that the current structure of the execution module of the holon is not suitable for the current environment. Therefore, structure adjustment will be triggered, decreasing the number of idle execution units, and these released resources will be used to support execution units who will be in state of high-load. Above all, in this case, the holon adjusts its inner structure to adapt to the environment. An example of structure adjustment is shown in FIGURE 8. FIGURE 8 (a) shows the Waiting Queue of a holon, and we can see that the tasks in the Waiting Queue require the functions *Func_X*, *Func_Y*, *Func_Z* and *Func_W*. Moreover, *Func_Z* requires 16.67% of the computing resources, as does *Func_Y*. *Func_W* needs 58.33% of the computing resources, while *Func_X* needs 8.33%. However, based on the current structure, the holon can only provide 25.67% of its computing resources for *Func_Z*, 25.67% for *Func_Y*, and 24.33% for *Func_W* and *Func_X*, as shown in FIGURE 8 (b). The difference between supply and demand causes a large accumulation of tasks for *Func_W*, while the computing resources for *Func_X* are idle enough. To cope with this change, the system triggers structure adjustments, as shown in FIGURE 8 (c). The holon decreases the number of subholons for *Func_X* and increases the number

Algorithm 1 Competitiveness Adjustment Algorithm

Input: the current competitiveness of a Holon $Competitiveness^{X}_{current}$; the waiting time $Task_i.Waiting\_time$, the expect time $Task_i.Expect\_time$, and the cost $Task_i.Req_{Func}.Cost$ of Task $Task_i$; the current competitiveness regarding subfunction y $Competitiveness\_Holon^{y\in X}_{current}$

Output: the current competitiveness of a Holon $Competitiveness^{X}_{current}$, the current competitiveness regarding subfunction y $Competitiveness\_Holon^{y\in X}_{current}$

If(a new task $Task_i$ is added to the Execution Queue)
then

update the current competitiveness of the Holon, and the current competitiveness regarding subfunction *y of the Holon*

$$Competitiveness\_Holon^{X}_{current} = Competitiveness\_Holon^{X}_{current} - \frac{Task_i.Waiting\_time}{Task_i.Expect\_time} \times Task_i.Req_{Func}.Cost.$$

$$Competitiveness\_Holon^{y\in X}_{current} = Competitiveness\_Holon^{y\in X}_{current} - \frac{Task_i.Waiting\_time}{Task_i.Expect\_time} \times Task_i.Req\_Func^{y}.Cost$$

If ($Task_i$ is deleted from the Execution Queue)
then

update the current competitiveness of the Holon,and the current competitiveness regarding subfunction *y of the Holon*

$$Competitiveness\_Holon^{X}_{current} = Competitiveness\_Holon^{X}_{current} + \frac{Task_i.Waiting\_time}{Task_i.Expect\_time} \times Task_i.Req_{Func}.Cost.$$

$$Competitiveness\_Holon^{y\in X}_{current} = Competitiveness\_Holon^{y\in X}_{current} + \frac{Task_i.Waiting\_time}{Task_i.Expect\_time} \times Task_i.Req\_Func^{y}.Cost$$

If $Task_i$ has been added to the Execution Queue, and is performing by subholon H, but Ti is not completed within the *Expect_time*, and no message about the task failure is received
then

Set the current competitiveness of subHolon H to 0
update the current competitiveness of the Holon, and the current competitiveness regarding subfunction *y of the Holon*

$$Competitiveness\_Holon^{X}_{current} = Competitiveness\_Holon^{X}_{current} - Competitiveness\_H^{X}$$

$$Competitiveness\_Holon^{y\in X}_{current} = Competitiveness\_Holon^{y\in X}_{current} - Competitiveness\_H^{y\in X}$$

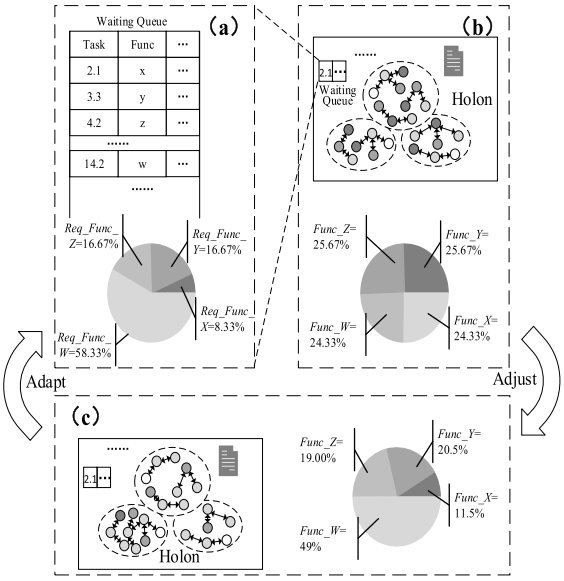

FIGURE 8. An example of structure adjustment when the waiting tasks have polarized.

of subholons for *Func_W* to adapt to the current environment. Finally, the proportion of subholons in the holon is *Func_W=49.00%, Func_X=11.50%, Func_Y=20.50%, and Func_Z=19.00%*; this proportion is basically consistent with the requirements of the Waiting Queue, realizing rational allocation of computing resources.

(2) Task failed. This situation can be further divided into two cases: 1) an execution unit failed to execute a task, but some of other execution units who are idle or working has enough competitiveness to execute the failed task. At this time, perhaps the current external environment is changes, results to the current execution unavailable. Then the inner self-adaptive mechanism is triggered. The Self-adaptive Adjustment Module will find some idle or working execution units to perform the task, the process is shown in FIGURE 9; 2) an execution unit failed to execute a task, and others execution units are in state of high-load or overload. In this case, computing resources of the current holon have reached the upper limit of utilization and cannot meet any task requirements. Therefore, inner structure adjustment for the current single holon is useless. To guarantee the normal

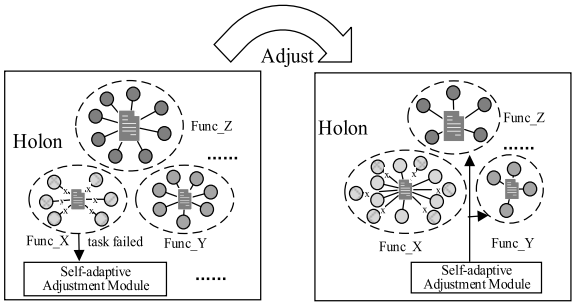

FIGURE 9. Process of inner structure adjustment of a Holon.

running of the system, other holons must assist the current holon. The Self-adaptive Adjustment Module in the holon finds that there is no computing resources to performed the failed task, therefore, it will request assistance from its parent, and then the self-adaptive mechanism of its parent holon will be triggered. At this time, the parent holon will find execution units who are idle or working in other subholons to perform the task. If there are no idle or working execution units in other subholons, then the self-adaptive mechanism of the ancestor will be triggered. The process of this feedback structure adjustment is shown in FIGURE 10.

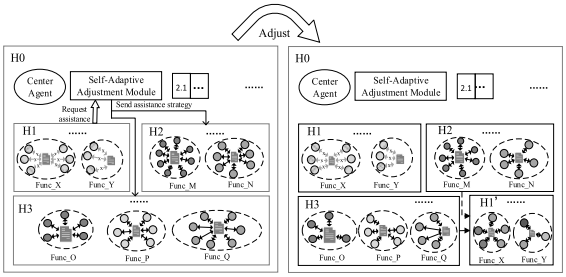

FIGURE 10. Process of outside structure adjustment of a Holon.

The algorithm of Self-Adaptive Mechanism is shown in ALGORITHM 2, the time complexity of this algorithm is *O(mn)*. If the inner self-adaptive mechanism is trigged, the membership of the execution units will not change, their parent Holon is the same. But if the outside self-adaptive mechanism is trigged, the membership of the execution units who provides assistance will get a new parent. When self-adaptive mechanism is triggered, the function map will be revised by the Self-Adaptive Adjustment Module, and the Task Assignment Module will reassign the tasks in the Waiting Queue of the execution units who need assistance, and then delete these tasks from the previous Waiting Queue. It is worth noting that if an execution unit is the overlap of different Holons, only the idle computing resources of the execution unit will be scheduled to provide assistance. For the high-load or overload execution units, the Holon marks them as ''unavailable'', and will not allow the execution units to participate task execution in a short time. However, after a certain period of time, the Holon will try to assign tasks to these ''unavailable'' execution units. Only if the execution units successfully execute the tasks 3 times in a row, the Holon will cancel their ''unavailable'' marks, and consider them as normal execution units.

C. TASK ASSIGNMENT MECHANISM

In this section, the task assignment mechanism for DHMAS is discussed. A heuristic search method based on competitiveness is proposed for the situation of a non-atomic parent holon assigning tasks for its sub-execution units, while the competitiveness -based contract net method is used for the situation of an atomic holon assigning tasks for its Function Agents.

Algorithm 2 Structure Adjustment Algorithm

```
If receives assistance request to assist H from the parent Holon
  prepare for structure adjustment // structure adjustment is triggered
  read the idle subholons set I and the working subholons set W;
  if(I ≠ ∅)
    for (m = 1;m <= I.size;m++)
    {
    Im is added to list L;
    If((SUM(Competitiveness_Li) > Task.cost)
    Break;
  else
      if(W ≠ ∅)
        for (n = 1;n <= W.size;n++)
      {
       If      Competitiveness_Wm_current > 0.5∗ Competitiveness_Wm)
         Wn is added to list L;
         If((SUM(Competitiveness _Li)> Task.cost)
           Break;
        else
           L.clear();
       }
     else
       L.clear();
     }
else{
          if( I= ∅&&W ≠ ∅)
            for (n=1;n<=W.size;n++)
         {
          If(   Competitiveness_Wm_current > 0.5∗ Competitiveness_Wm)
            Wn is added to list L;
            If((SUM(Competitiveness _Li)> Task.cost)
              break;
            else
              L.clear();
          }
        else{ L.clear();}
     }
  If(L.isEmpty())
          feedback to the parent that assistance failed
  else
          all execution units in L will be scheduled to assist the execution units;
          L.clear();
}
If a task executed failed
     prepare for structure adjustment//structure adjustment is triggered
     read the idle Execution Units set E and the working Execution Units set U;
          if(E ≠ ∅)
          then
```

Algorithm 2 *(Continued.)* Structure Adjustment Algorithm

```
            for (m = 1;m <= E.size;m++)
        {
         Em is added to list L;
         If((SUM(Competitiveness _Li)> Task.cost)
          break;
         else{
                  if(U ≠ ∅)
                      for (n = 1;n <= U.size;n++)
              {
                If(Competitiveness_Wm_current > 0.5∗
                                  Competitiveness_Wm)
                  Un is added to list L;
                  If((SUM(Competitiveness _Li)>
Task.cost)                 break;
                  else
                      L.clear();
                }
              else
                  L.clear();
            }
          }
    else{if(E = ∅&&U ≠ ∅)
            for (n = 1;n <= U.size;n++)
             {If(Competitiveness_Wm_current > 0.5∗
                                  Competitiveness_Wm)
              Un is added to list L;
              If((SUM(Competitiveness_Li)> Task.cost)
                break;
              else
                L.clear();
            }
            else{L.clear();}
          }
     If(L.isEmpty())
     feedback to the parent that assistance failed
     else
       all execution units in L will be scheduled to assist the
execution units;
       L.clear();
    }
If f ∈ Task.Req_Fuc && f.Cost / Task.Cost > 0.5 &&
(Competitiveness_Holon^f_current ≠ Max_{i∈Task.Req_Fuc}
Competitiveness^i_current)
     prepare for structure adjustment//structure adjustment
is triggered
     read the low-cost execution units set P;
     if(P ≠ ∅)
         for(q = 1;q <= P.size;q++)
          {
         Pi is added to list L;
         If(Competitiveness_Li+Competitiveness_
Holon^f_current) > 0.5 ∗ Competitiveness_Holon_current)
```

Algorithm 2 *(Continued.)* Structure Adjustment Algorithm

```
            break;
          else
            L.clear()
        }
     else{
       L.clear();
     }
     If(L.isEmpty())
     feedback to the parent that assistance failed
     else
   all execution units in L will be scheduled to assist the
execution units who provides f
       L.clear();
}
```

1) HEURISTIC SEARCH METHOD BASED ON COMPETITIVENESS

The heuristic search method based on competitiveness takes a top-down perspective. There is a hierarchical relationship between the task assigner and the task performer, that is, the task assigner at the higher level commands the task performer at the lower level to complete the task. Because the selection of a suitable task performer is not a simple problem, the task assignment module needs to use a reasonable search strategy to find an appropriate task performer to speed up the search process. A reasonable search strategy will improve the system efficiency.

In the proposed model, the Center Agent plays the role of task assigner, while the execution units (subholons) in the Execution Module are the task performers. But the task assignment strategy is generated by the task assignment module. After receiving tasks allocated by the parent holon, the Center Agent decomposes the tasks into several subtasks that can be executed by the subholons at the lower layer, and the Task Assigner Module generates a reasonable task assignment strategy based on the related competitiveness of execution units. Because the arriving task could be divided into several different types of subtasks, and a type of subtask will be performed by a specific type of subholon, each type of subtask needs to find some suitable execution units to do. What's more, for a certain type of task, there will be multiple subholons that can be completed, for each type of sub-task assignment, there should be an assignment strategy.

As a random searching algorithm, the simulated annealing algorithm is suitable for solving large-scale combinatorial optimization problems, and its calculation process is relatively simple. What's more, the simulated annealing algorithm has the ability to find a global optimal solution. However, this algorithm is sensitive to the selection of parameters. This article utilizes an improved simulated annealing

algorithm to generate a task assignment strategy when the parent Holon assigns tasks to its subHolons. Three factors determine the adaptability of a subholon, which is the objective function of the search process:

1). The total competitiveness of the subholon, written as $Competitiveness\_Holon^X$;

2). The current competitiveness of the subholon, written as $Competitiveness\_Holon^X_{current}$;

3). The current competitiveness of the subholon for function y (yϵX), written as $Competitiveness\_Holon^{y\epsilon X}_{current}$.

The heuristic search method based on competitiveness searches for a sub-holon that will achieve maximum adaptability as the task performer. The adaptability is calculated by (10).

$$Adaptability^X = \frac{Competitiveness\_Holon^X_{current}}{Competitiveness\_Holon^X} \times Competitiveness\_Holon^{y\epsilon X}_{current} \quad (10)$$

If there are p subtasks waiting to be executed, and q sunholons could perform this type of subtasks (p<q), then the size of search space is C^{q-1}_{p-1}, then the space complexity of the algorithm is $O(C^{q-1}_{p-1})$.

To improve the quality of the solution, when we initialize the temperature of the simulated annealing algorithm, some steps of heating are added. The procedure of the simulated annealing algorithm is as follows:

Step 1: Determine the objective function $f(x)$ and the search space, randomly generate an initial solution x0 from the solution space;

Step 2: Set the initial temperature $t_0 = 0$, and, choose a new solution x_1 from the neighborhoods of x_0, if the value of the objective function increases, set $x_0 = x_1$, and set $T_0 = h(t_0)$, $h(t_0)$ is an increasing function.

Step 3: If the number of heating has reached n, then go to step4, else, go to step 2;

Step 4: determine the function of decreasing temperature $f(t)$;

Step 5: Take T_0 and x_0 as the initial temperature of annealing and initial solution. If the inner loop has been executed k times, go to step 6; else, randomly select a new solution x_j from the neighborhoods $N(x_i)$ and calculate the value of the objective function. If the value increases, then x will be accepted, repeat step5;

Step 6: Repeat step 5 until the number of inner loops reached t, record the maximum value of the objective function and the corresponding solution

Step 7: Set $t_{n+1} = f(t_n), n = n + 1$; if the loop time has reached l, then the search process ends, then else go to step 5.

Then the time complexity of the improved simulated annealing algorithm is $O(tl)$.

2) COMPETITIVENESS-BASED CONTRACT NET METHOD

The Contract Net Protocol, proposed by Davis and Smith [31], imitates the bidding mechanism in human society. When an agent receives a task that cannot be completed independently, it will broadcast the task demands to other agents and start a bidding process. After receiving the broadcast, other agents determine whether to participate in the bidding according to the task requirements and their own competitiveness. By analyzing each bidder, the biddee selects the appropriate bidder according to the current status of all bidders and establishes cooperation with the selected bidder.

An atomic holon (that is, the holon is indivisible) is composed of a Center Agent and several Function Agents.There are no subholons in the Execution Module of an atomic holon. The function of a Function Agent is relatively simple, the allocation and handover of tasks are frequent. Moreover, there is a large number of Function Agents, and these agents are loosely distributed. Therefore, the contract net protocol is adopted in an atomic holon. It is beneficial to make full use of the characteristic of intelligence of an agent to improve the efficiency of task assignment.

There are two shortcomings in the traditional contract network protocol: (1) The communication is carried out by means of broadcasting, which increases communication overhead of the system. (2) Only the static state of agents are considered, and ignore the state of each agent will change as the system running. Therefore, this article improves the traditional contract net protocol, takes advantage of a blackboard instead of the method of broadcast communication, and each agent bids for task based on its current competitiveness. In the Task Assignment Module of an atomic holon, there is a public blackboard that all Function Agents can access. The blackboard is divided into different partitions to record the different functional tasks. The Function Agents wait for the tasks to be published in the corresponding partition according to the functions they can provide and participate the bidding according to their current states. Because each task has its own *Expect_time*, the Center Agent does not wait for all Function Agents to complete the bid, but collects the bids before a certain time point, and analyzes the bidders to find the most appropriate performers and then assigns the task to the corresponding Function Agents. Our improved method avoids some unnecessary communications, and considers the real-time competitiveness of each agent. The process of task assignment is shown in FIGURE 11.

From FIGURE 11, Step ① represents the Center Agent inviting bids. The Center Agent selects a specific task from the Waiting Queue, publishes the task to the specific partition on the blackboard according to the required function of the task, and then waits for bidders. Step ② represents the process of bidding by the Function Agents. A Function Agent obtains the bidding information on the corresponding partition of the blackboard, bids according to its own load state, and waits for feedback from the Center Agent. Meanwhile, the Function Agent still performs the current tasks synchronously. Step ③ is the process by which the Center Agent acquires the bid information of the Function Agents. After analysis, the Center Agent selects the appropriate Function Agents and assigns the task to the Function Agent with the winning bid, as shown in Step ④.

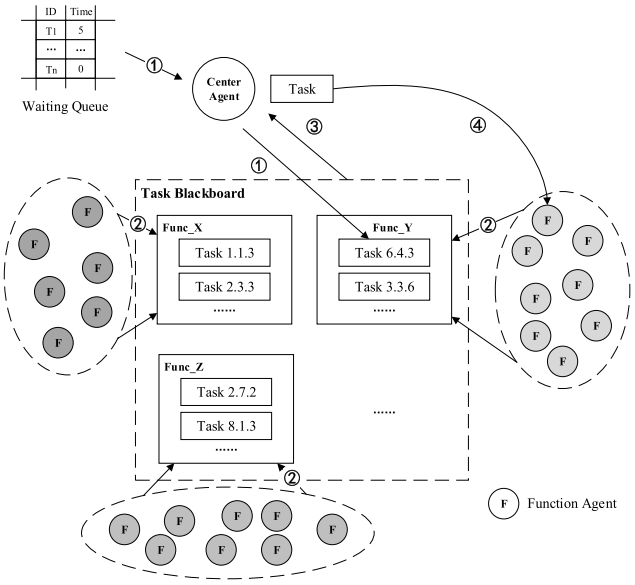

FIGURE 11. Process of the Competitiveness -based contract net method.

The Function Agents participate in the bidding according to their own competitiveness, which are related to the numbers of tasks in their Waiting Queues. The current competitiveness of a Function Agent is calculated by (11):

$$\begin{aligned} &Competitiveness\_Agent^{X}_{current} \\ &= Competitiveness\_Agent^{X} - \sum_{i=1}^{n} Load^{Task_i} \\ &= Competitiveness\_Agent^{X} - \sum\nolimits_{i=1}^{n} (\frac{Task_i.Waiting\_time}{Task_i.Except\_time} \\ &\quad \times Task_i.Req\_func.Cost) \end{aligned} \tag{11}$$

In addition, when a Function Agent fails to perform a certain task, it will no longer actively participate in the bidding to avoid increasing its task load. After collecting the bids of all Function Agents, the Center Agent assigns the task according to the current competitiveness of the Function Agents.

IV. CASE STUDY AND EXPERIMENTAL RESULTS

Metasearch engine integrates search results from multiple underlying search engines to improve recall ratio in the big data environment. Because metasearch engine relies on the results of component search engines, the states of component search engines affect the system running, that is, metasearch engine is more sensitive to the environment. Furthermore, for a personalized metasearch engine, it must record user behaviors and analyze user preferences when a large number of users log in, and quickly return search results to users. Therefore, personalized metasearch engine runs in a dynamic environment, and must process a large number of complex tasks. This section introduces a personalized metasearch engine, named IM Search, which is based on our proposed DHMAS. IM Search is a metasearch engine that combines the search engines "Youdao," "Baidu," "Bing," "Yahoo," and "Sogou." By analyzing users' click histories, IM Search has the ability to obtain user preferences and provide personalized services, including personalized ranking and information recommendation.

A. ORGANIZATIONAL STRUCTURE FOR IM SEARCH

To achieve environmental requirements, IM Search must implemented 15 atomic functions as followings:

(1) Schedule search engine Baidu to get search results;
(2) Schedule search engine Youdao to get search results;
(3) Schedule search engine Bing to get search results;
(4) Schedule search engine Yahoo to get search results;
(5) Schedule search engine Sogou to get search results;
(6) Record user click histories;
(7) Remove the duplicate results returned by component search engines;
(8) Analyze all the topics of the webpage;
(9) Analyze user interests based on user click histories;
(10) Rank all the returned webpages into a single list;
(11) Cluster users into different groups based on the clicked results;
(12) Analyze user intention based on the group relationship;
(13) Recommend webpages based on user intention;
(14) Recommend querywords based on user intention;
(15) Display the search results, the recommended querywords and webpages to user.

Corresponding to the above functions, IM Search must contain 15 roles, Baidu Role, Youdao Role, Bing Role, Yahoo Role, Sogou Role, Record Role, Deduplication Role, Topic Role, Analysis Role, Sort Role, Group Role, Intention Role, Webpage Role, Queryword Role, and Display Role. In the initial state of the system, each role is performed by an agent at least. During the running time of the system, roles and agents are dynamic binding. As the environment changes, an agent will play different roles to meet the environmental requirements, which caused the changes of the structure of the system. The holonfication is based on the data flow graph. Therefore, we analyzed the data flow between different agents, as shown in FIGURE 12.

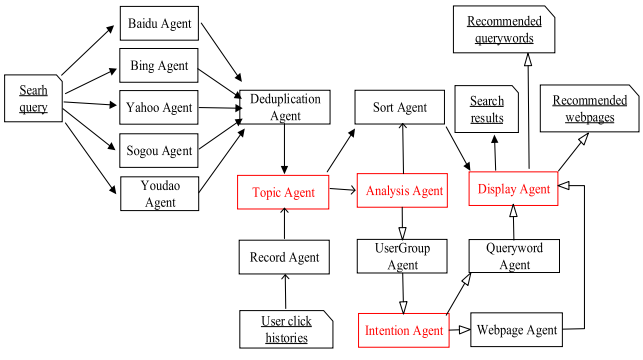

FIGURE 12. Data flow of IM Search.

There are two input data received by IM Search. After processing these input data, IM Search generates three output data. From FIGURE 12, we can see that the Topic Agent, Analysis Agent, Intention Agent, and Display Agent are the diving lines. The reason is that, after processed by these agents, the data will flow to multiple other agents. For example, when the data are processed by Topic Agent, the processed data will further flow to the Analysis Agent or the Sort Agent. The Topic Agent could be regarded as a diving line. Then according to the diving line and the data flow, the Baidu Agent, Bing Agent, Yahoo Agent, Sogou Agent, Youdao Agent, Deduplication Agent and the Topic Agent form a Holon, named ResultsPreprocess Holon, which provides function of preprocessing the search results. Similarly, the initial structure of IM Search will be organized, as shown in FIGURE 13.

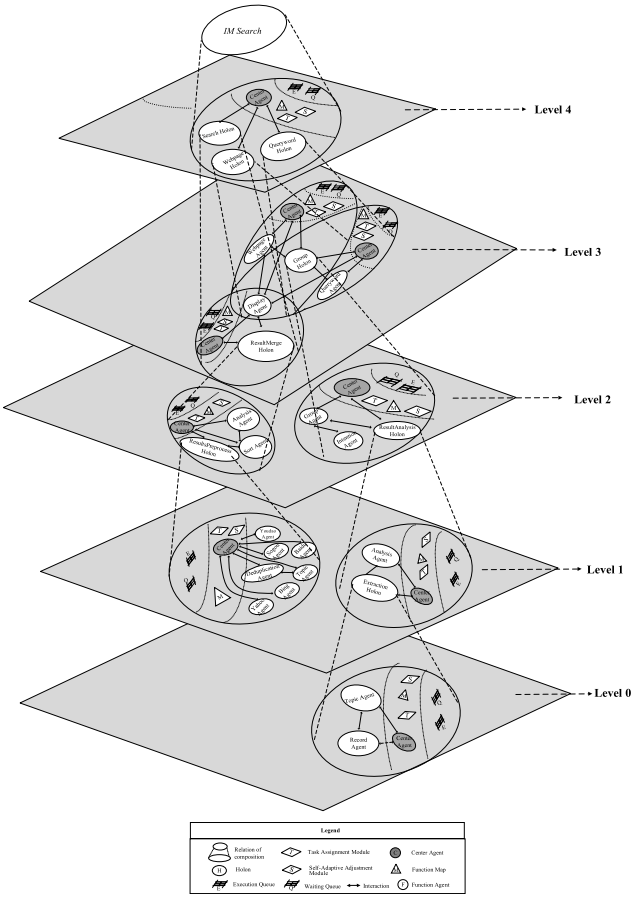

FIGURE 13. Organizational structure of IM Search based on DHMAS.

IM Search consists of 5 levels:

(1) Level 4: This level is the top level, consists of a Center Agent, a Waiting Queue, an Execution Queue, a Function Map, a Self-Adaptive Adjustment Module, a Task Assignment Module, some Search Holons, some Webpage Holons, and some Queryword Holons. The Center Agent analyzes the tasks requirements, decomposes the tasks further into subtasks, and assigns the subtasks to the Search Holon, Webpage Holon or Queryword Holon according to the task assignment strategy generated by the Task Assignment Module. The Search Holon performs the task of returning the personalized search results to a user. The Webpage Holon and the Queryword Holon perform the tasks of providing the recommended webpages and the recommended querywords to a user, respectively.

(2) Level 3: This level demonstrates the structures of the Search Holon, the Webpage Holon, and the Queryword Holon. The Search Holon contains a Center Agent, a Waiting Queue, an Execution Queue, a Function Map, a Self-Adaptive Adjustment Module, a Task Assignment Module, some ResultMerge Holons, and some Display Agents. The Center Agent of the Search Holon analyzes the tasks requirements, decomposes the tasks further into subtasks, and assigns the subtasks to the ResultMerge Holons and the Display Agents, according to the task assignment strategy generated by the Task Assignment Module. The ResultMerge Holon performs the task of merging the returned search results based on use interests. The Display Agent performs the task of displaying the result list to a user. The Webpage Holon contains a Center Agent, a Waiting Queue, an Execution Queue, a Function Map, a Self-Adaptive Adjustment Module, a Task Assignment Module, some Group Holons, some Display Agents, and some Webpage Agents. The Center Agent of the Webpage Holon analyzes the tasks requirements, decomposes the tasks further into subtasks, and assigns the subtasks to the Group Holons, the Webpage Agents, and the Display Agent, according to the task assignment strategy generated by the Task Assignment Module. The Group Holon performs the task of obtaining user intention based on group members. The Webpage Agent performs the task of recommending webpages based on user intention, while the Display Agent performs the task of displaying the recommended webpages to a user. The structure of Queryword Holon is similar to that of webpage Holon. Therefore, we will not go into details.

(3) Level 2: This level shows the structures of the ResultMerge Holon, and the Group Holon. The ResultMerge Holon contains a Center Agent, a Waiting Queue, an Execution Queue, a Function Map, a Self-Adaptive Adjustment Module, a Task Assignment Module, some Analysis Agents, and some ResultsPreprocess Holons. The Center Agent of the ResultMerge Holon analyzes the tasks requirements, decomposes the tasks further into subtasks, and assigns the subtasks to the Analysis Agents, the sort Agents, and the ResultPreprocess Holons, according to the task assignment strategy generated by the Task Assignment Module. The Analysis Agent performs the task of obtaining user interests from the topics extracted from the webpages. The ResultPreprocess Holon performs the task of

preprocessing the returned results returned by the component search engines. The Sort Agent performs the task of ranking all the returned webpages into a single list. The Group Holon contains a Center Agent, a Waiting Queue, an Execution Queue, a Function Map, a Self-Adaptive Adjustment Module, a Task Assignment Module, some ResultAnalysis Holons, some Group Agents, and some Intention Agents. The Center Agent of the Group Holon analyzes the tasks requirements, decomposes the tasks further into subtasks, and assigns the subtasks to the ResultAnalysis Holons, the Group Agents, and the Intention Agents, according to the task assignment strategy generated by the Task Assignment Module. The Group Agent performs the task of clustering users into different groups. The ResultAnalysis Holon performs the task of analyzing the topics to which the webpage belongs. The Intention Agent performs the tasks of analyzing user intention based on the group relationship that the Group Agents provides;

(4) Level 1: The structures of the ResultsPreprocess Holon and the ResultAnalysis Holon are shown in this Level. The ResultsPreprocess Holon contains a Center Agent, a Waiting Queue, an Execution Queue, a Function Map, a Self-Adaptive Adjustment Module, a Task Assignment Module, some Baidu Agents, some Bing Agents, some Sogou Agents, some Youdao Agents, some Yahoo Agents, some Deduplication Agents, and some Topic Agents. The Center Agent of the ResultsPreprocess Holon analyzes the tasks requirements, decomposes the tasks further into subtasks, and assigns the subtasks to the Baidu Agents, the Bing Agents, the Sogou Agents, the Youdao Agents, the Yahoo Agents, the Deduplication Agents, and the Topic Agents, according to the task assignment strategy generated by the Task Assignment Module. As the execution unit, the Baidu Agent performs the task of scheduling search engine Baidu to get search results. Similarly, the Youdao Agent, the Bing Agent, the Sogou Agent and the Yahoo Agent perform the tasks of scheduling search engine Youdao, Bing, Sougo and Yahoo to get search results, respectively. The Deduplication Agent performs the task of removing the duplicate results returned by component search engines, while the Topic Agent performs the task of analyzing all the topics of the webpage. The ResultAnalysis Holon contains a Center Agent, a Waiting Queue, an Execution Queue, a Function Map, a Self-Adaptive Adjustment Module, a Task Assignment Module, some Analysis Agents, and some Extraction Holons. The Center Agent of the ResultAnalysis Holon analyzes the tasks requirements, decomposes the tasks further into subtasks, and assigns the subtasks to the Analysis Agents and the Extraction Holons, according to the task assignment strategy generated by the Task Assignment Module. The Analysis Agent performs the task of analyzing the topics to which the returned webpage belongs. The Extraction Holon perform the task of extracting the topics from the webpages.

(5) Level 0: This level shows the structures of the Extraction Holon. The Extraction Holon contains a Center Agent, a Waiting Queue, an Execution Queue, a Function Map, a Self-Adaptive Adjustment Module, a Task Assignment Module, some Record Agents, and some Topic Holons. The Center Agent of the Extraction Holon analyzes the tasks requirements, decomposes the tasks further into subtasks, and assigns the subtasks to the Record Agents and the Topic Agents, according to the task assignment strategy generated by the Task Assignment Module. The Record Agent performs the task of recording the user click histories. The Topic Agent performs the task of analyzing all the topics of the webpage.

The holons which are located at Level 1 utilize the Competitiveness-based contract net method to assign tasks to the Function Agents in Level 0. The heuristic search method based on competitiveness is utilized when other high-level Holons assign tasks to low-level execution units, such as, when the holons in Level 2 assign tasks to the holons or agents in Level 1, and so on.

The homepage and the returned results page of IM Search are shown in Figures 14 and 15. The methods for user interest acquisition and result merging method are described in our previous works [32]. The details of clustering users into different groups and generating recommendation can be found in [33]. User intention analysis is discussed in [34], [35].

FIGURE 14. Homepage of IM Search.

B. DEFINITION OF FUNCTION AND COMPETITIVENESS

There are 8 types of Holons, and 16 types of Agents in IM Search. The function, the cost and the Competitiveness of each agent is shown in TABLE 2.

For a Holon, the competitiveness is directly related to its structure and function it provides. The function of a Holon is a composite function. Therefore, the function, the cost and the competitiveness of each Holon is shown in TABLE 3.

FIGURE 15. Returned results page of IM Search.

It is worth noting that the parameters (i.e. *m, n*, ...) change dynamically during system running, because the structure adjustment mechanism may be triggered to make the system adapt to the environment, such as when different numbers of users utilize IM Search to query or user requirements change, and so on. Therefore, the self-adaptive mechanism of the system makes the settings of parameters affect the efficiency of task execution only in a small range.

C. REPRESENTATION OF TASK

The operations performed by users are generally divided into five types: ①. Users log in and request all component search engines; ②. Users log in and request parts of component search engines; ③. Users do not log in but request all component search engines; ④. Users modify their personal information; ⑤. Users do not log in but request parts of component search engines.

Take the scenario where users log in and request all component search engines for example:

IM Search receives requirement from the environment:

Requirement::=< Search_all, webpage_recommendation, queryword_recommendation >;

The Center Agent at level 2 recognizes the task and decomposes the task into 3 subtasks based on this requirement:

Task:: =<1, Func_SE, 0.3, Waitting, Ø>
Task:: =<2, Func_WP, 0.3, Waitting, Ø>
Task:: =<3, Func_QW, 0.3, Waitting, Ø>
Task 1 is further decomposed into 4 subtasks:
Task:: =<1.1, Func_RM, 0.2, Waitting, 1.2>
Task:: =<1.2, func_DI, 0.1, Waitting, Ø>
Task 1.1 is further decomposed into 3 subtasks:
Task:: =<1.1.1, Func_RP, 0.1, Waitting, 1.1.2>
Task:: =<1.1.2, func_AA ,0.15, Waitting,1.1.3>
Task:: =<1.1.3, func_SO, 0.05, Waitting, Ø>
Task 1.1.1 is further decomposed into 7 subtasks:
Task:: =<1.1.1.1, func_SO, 0.02, Waitting, 1.1.1.6>
Task:: =<1.1.1.2, func_BA, 0.02, Waitting, 1.1.1.6>
Task:: =<1.1.1.3, func_BI, 0.02, Waitting, 1.1.1.6>
Task:: =<1.1.1.4, func_YO, 0.02, Waitting, 1.1.1.6>

TABLE 2. Function, cost and competitiveness of each agent (r refers to the average resource per unit).

Agent	Function	Cost	Competitiveness
Center Agent	Providing the way of communication, analyzing task requirements, decomposed the task, and assigning the subtasks.	*1r*	*1r*
Sort Agent	Ranking all the returned webpages into a single list, written as *func_SO*	*3r*	*3r*
Group Agent	Clustering users into different groups based on the clicked results, written as *func_GA*,	*3r*	*3r*
Record Agent	Recording user click histories, written as *func_RA*	*1r*	*1r*
Topic Agent	Analyzing all the topics of the webpage, written as, *func_TA*	*3r*	*3r*
Analysis Agent	Analyzing user interests based on user click histories, written as *func_AA*	*3r*	*3r*
Webpage Agent	Recommending webpages based on user intention, written as *func_WP*	*3r*	*3r*
QueryWord Agent	Recommending querywords based on user intention, written as *func_QW.*	*3r*	*3r*
Intention Agent	Analyzing user intention based on the group relationship, written as *func_IN*	*3r*	*3r*
Display Agent	Displaying the search results, the recommended querywords and webpages to user, written as *func_DI.*	*1r*	*1r*
Sougo Agent	Scheduling the search engine Sougo, and maintains the interface of Sougo, written as *func_SO.*	*2r*	*2r*
Bing Agent	Scheduling the search engine Bing, and maintaining the interface of Bing, written as *func_BI.*	*2r*	*2r*
Yahoo Agent	Scheduling the search engine Yahoo, and maintaining the interface of Yahoo, written as *func_YA.*	*2r*	*2r*
Baidu Agent	Scheduling the search engine Baidu, and maintaining the interface of Baidu, written as *func_BA.*	*2r*	*2r*
Youdao Agent	Scheduling the search engine Youdao, and maintaining the interface of Youdao, written as *func_YO.*	*2r*	*2r*
Deduplication Agent	Removing the duplicate results returned by component search engines; , written as *func_DE.*	*3r*	*3r*

TABLE 3. Function, cost and competitiveness of each Holon (r refers to the average resource per unit).

Holon	Function	Structure (besides Center Agent)	Cost	Competitiv_ eness
Result_ Preprocess Holon	Preprocessing the returned results returned by the component search engines, written as *Func_RP={func_ TA, func_DE, func_SO, func_BA, func_BI, func_YO, func_YA }.*	*m* Topic Agents, and *n* Deduplication Agents, *h* Baidu Agents, *b* Bing Agents, *c* Sougo Agents, *d* Youdao Agents, and *e* Yahoo Agents	*17r*	*[3(m+n)+2(h +b+c+d+e)+ 1]r*
ResultMe_ rge Holon	Merging the returned search results based on use interests, written as *Func_RM={func_ AA, Fun_RP, fun_SO}*	*p* Analysis Agents, *q* ResultPreproc ess Holon, and *a* Sort Agents	*24r*	*[3(a+p)+3q(m+n)+2q(h+ b+c+d+e)+q +1]r*
Search Holon	Returning the search results to a user, written as *Func_SE={Func_ RM, func_DI }*	*i* ResultMerge Holon, and *j* Display Agent	*26r*	*[[3i(a+p)+3qi (m+n)+2qi(h +b+c+d+e)+ qi+i]+j+1]r*
QueryWo_ rd Holon	Providing the recommended query words, written as *Func_QW={func_ QW , Func_GH, func_DI }*	*f* Queryword Agents, *o* Group Holons, and *g* Display Agents	*21r*	*[3f+o x[w(3k+l)+3t +1]+3y+3z+ 1]+g+1]r*
Webpage_ Holon	Providing the recommended webpages, written as *Func_WP={ func_Wp , Func_GH, func_DI }*	*m* Webpage Agents, *n* Group Holons, and *s* Display Agents	*21r*	*3m+n[w(3k+l)+3t+1]+3y+ 3z+1]+t+1]r*
ResultAn_ alysis Holon	Analyzing the topics to which the webpage belongs, written as *Func_RA={Func_ EX, func_AA}*	*w* Extraction Holons , and *t* Analysis Agents	*9r*	*[w(3k+l)+3t+ 1]r*
Group_ Holon	Obtaining user intention based on group members, written as *Func_GH={Func _RA, fun_IN, func_GA}.*	*x* ResultAnalysi s Holons, *y* Intention Agents, and *z* Group Agents	*16r*	*x[w(3k+l)+3t +1]+3y+3z+ 1]r*
Extracti_ on Holon	Extracting the topics from the webpages, written as *Func_EX={func_ TA, func_RA}.*	*k* Topic Agents, and *l* Record Agents	*5r*	*(3k+l)r*

Task:: =<1.1.1.5, func_YA, 0.02, Waitting, 1.1.1.6>
Task:: =<1.1.1.6, func_DE, 0.04, Waitting, 1.1.1.7>
Task:: =<1.1.1.7, func_TA, 0.04, Waitting, Ø>
Task 2 is further decomposed into 3 subtasks:
Task:: =<2.1, Func_GH, 0.12, Waitting, 2.1.2>
Task:: =<2.2, func_WP, 0.08, Waitting, 2.1.3>
Task:: =<2.3, func_DI, 0.1, Waitting, Ø>
Task 2.1 is further decomposed into3 subtasks:
Task:: =<2.2.1, Func_RA, 0.04, Waitting, 2.2.2>
Task:: =<2.2.2, func_GA, 0.04, Waitting, 2.2.3>
Task:: =<2.2.3, func_IN, 0.04, Waitting, Ø>
Task 2.2.1 is further decomposed into 2 subtasks:
Task:: =<2.2.1.1, Func_EX, 0.03, Waitting, 2.2.1.2>
Task:: =<2.2.1.2, func_AA, 0.01, Waitting, Ø >
Task 2.2.1.1 is further decomposed into 2 subtasks:
Task:: =<2.2.1.1.1, func_RA, 0.01, Waitting, 2.2.1.1.2>
Task:: =<2.2.1.1.2, func_ TA, 0.02, Waitting, Ø >
Task 3 is further decomposed into 3 subtasks:
Task:: =<3.1, Func_GH, 0.12, Waitting, 3.1.2>
Task:: =<3.2, func_QW, 0.08, Waitting, 3.1.3>
Task:: =<3.3, func_DI, 0.1, Waitting, Ø>
Task 3.1 is further decomposed into 2 subtasks:
Task:: =<3.2.1, Func_RA, 0.04, Waitting, 3.2.2>
Task:: =<3.2.2, func_GA, 0.04, Waitting, 3.2.3>
Task:: =<3.2.3, func_IN, 0.04, Waitting, Ø>
Task 3.2.1 is further decomposed into 2 subtasks:
Task:: =<3.2.1.1, Func_EX, 0.03, Waitting, 3.2.1.2>
Task:: =<3.2.1.2, func_AA, 0.01, Waitting, Ø >
Task 3.2.1.1 is further decomposed into 2 subtasks:
Task:: =<3.2.1.1.1, func_RA, 0.01, Waitting, 3.2.1.1.2>
Task:: =<3.2.1.1.2, func_ TA, 0.02, Waitting, Ø >

D. EXPERIMENTAL RESULTS OF SELF-ADAPTIVE MECHANISM

The experiments are designed from two aspects to verify the Self-adaptive Mechanism of IM Search.

1) EXPERIMENTAL RESULTS OF COMPETITIVENESS ADJUSTMENT

To test whether IM Search can adjust the competitiveness of each Holon when executing tasks, we invited 3 users to request queries using IM Search. Then the competitiveness of parts of Holons is recorded when 0 user, 1 user and 3 users initiate queries. The results are shown in Table 4. It is obvious that, as users initiate queries, the competitiveness of all listed

TABLE 4. Competitiveness of parts of Holons when different numbers of users request queries.

Holon	0 user	1 user	3 users
ResultAnalysis Holon	*17r*	*11r*	*11r*
ResultMerge Holon	*113r*	*87r*	*67r*
Search Holon	*148r*	*100r*	*99r*
QueryWord Holon	*61r*	*41r*	*21r*
Webpage Holon	*61r*	*41r*	*21r*
ResultPreprocess Holon	*39r*	*36r*	*30r*
Extraction Holon	*13r*	*7r*	*7r*
Group Holon	*32r*	*16r*	*16r*

Holons are decreasing in general, because the execution of tasks consumes the competitiveness of the related execution units.

2) EXPERIMENTAL RESULTS OF STRUCTURE ADJUSTMENT

To test the structure adjustment of IM Search, the test tool Jminer is utilized to generate a specific number of threads to access IM Search to simulate multiple users requesting queries.

When users log in IM Search, IM Search provides the returned the search results, webpage recommendation, and query words recommendation to users. While, users do not log in IM Search, IM Search only provides the search results to users. To test whether the system can trigger the structure adjustment mechanism when the waiting tasks are polarized, Jminer is utilized to simulate 5 login users utilize IM Search to initiate queries. After a period of time, increase the number of users to 10, and then increase the number of users to 100 again. The latest 90 users are asked to request queries in non-login status. To analyze the structure of IM Search In these three cases, the numbers of different execution units in Level 3 are recorded, as shown in FIGURE 16. We can see that, when the numbers of users are 5 and 10, the numbers of different execution units has not changed. The reason is that the competitiveness of each execution unit is enough to ensure the successful completion of the tasks, and the waiting tasks are not polarized. However, when the number of users is 100, and 90 of these users do not log in IM Search, the numbers of different execution units changed. This means the structure adjustment mechanism is triggered. Because when these 90 users request queries, IM Search only receives a large number of tasks of merging result into a single list. The waiting tasks have been polarized, and IM Search needs more competitiveness to return the search results to a user. Then the number of ResultMerge Holon increases, but the numbers of Webpage Agent, Queryword Agent, and Group Holon decrease. A part of computing resources of Webpage Agent, Queryword Agent, and Group Holon are given to ResultMerge Holon.

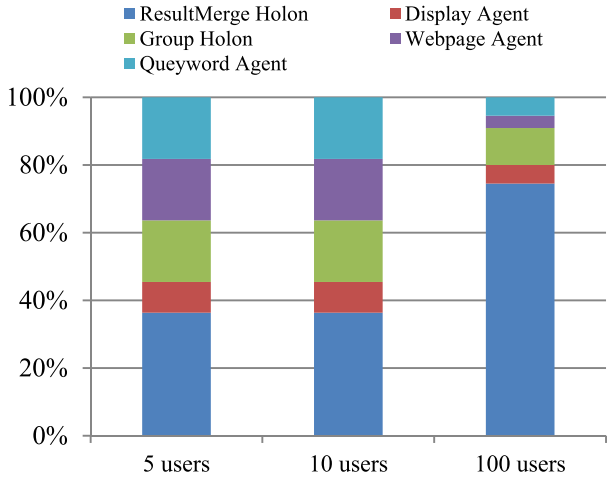

FIGURE 16. Experimental results of structure adjustments when the waiting tasks has been polarized.

IM Search is composed of 5 component search engines, users could selectively schedule parts of component search engines to search. Jminer is utilized to simulate 300 users request queries, and the component search engines are limited to Baidu, Bing, and Sougo. After a period of time, only 100 users are asked to request queries, and the component search engines are limited to Baidu, Bing, Sougo Youdao, and Yahoo. Then the structure of ResultPreprocess Holons in IM Search is analyzed, the experimental results are shown in FIGURE 17. We can see that when 300 users schedule Baidu, Bing and Sougo, there are no Yahoo Agents and Youdao Agents. However, when 100 users select Baidu, Bing, Sougo, Yahoo and Youdao to request queries, there are Baidu Agents, Bing Agents, Sogou Agent, Yahoo Agents and Youdao Agents in the ResultProcess Holon. This means, the structure adjustment mechanism is triggered. When users schedule Youdao search engine and Yahoo search engine, but there are no Yahoo Agents and Youdao Agent. The task of scheduling Yahoo and Youdao are performed failed. Therefore, the structure adjustment mechanism reallocates some computing resources to Yahoo Agents and Youdao Agents to ensure that the tasks can be executed successfully. Furthermore, it is obvious that the numbers of agents in ResultMerge Holon when users schedule Baidu, Bing, Sougo, Yahoo and Youdao is the same as that of when users schedule Baidu, Bing, and Sougo. This indicates that the inner structure adjustment mechanism of the ResultMerge Holon is triggered.

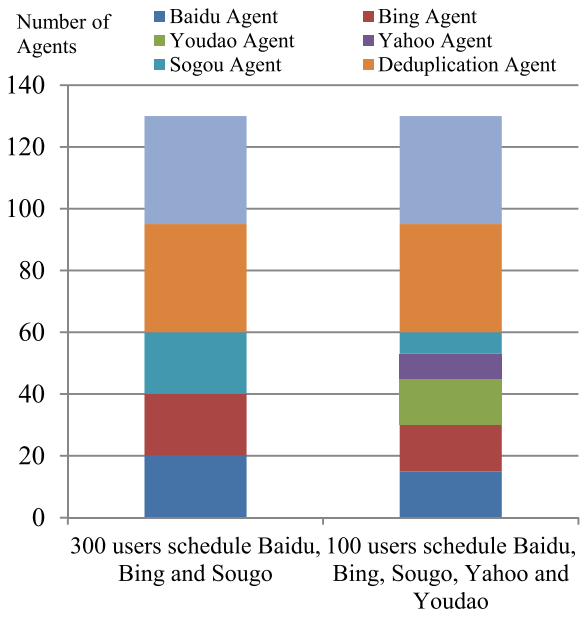

FIGURE 17. Experimental results of inner structure adjustment.

To test whether the outside structure adjustment mechanism can be triggered, Jminer is utilized to simulate 300 users request queries in login status, and the component search

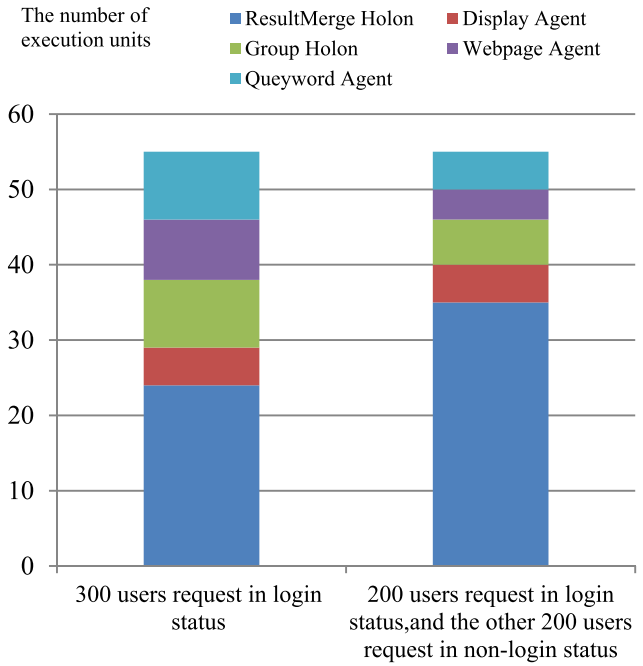

FIGURE 18. Experimental results of outside structure adjustment.

engines are limited to Baidu, Bing, and Sougo. After a period of time, we increase the number of users to 400, and 200 users are asked to request queries in login status, while the other 200 user are invited to request in non-login status. Then the number of ResultMerge Holon, Display Agents, Webpage Agents, Webpage Agents, and Group Holons are recorded. The experimental results are shown in FIGiURE 18. From the figure, we can see that, although the Group Holon, the Webpage Agent, the Queryword Agent, and the ResultMerge Holon do not belong to the same parent Holon. But the Group Holon, the Webpage Agent, and the Queryword Agent gives some computing resources to the ResultMerge Holon, to cope with the increasing search pressure which caused by the increase in the number of users. It indicates that the outside structure adjustment mechanism is triggered.

E. EXPERIMENTAL RESULTS OF SUCCESS RATE AND RESPONSE TIME

Meanwhile, we also implemented a personalized meta-search engine according to the component-based method, named MetSearch, whose algorithms are the same with IM Search. MetSearch also employs ''Baidu'', ''Youdao'', ''Yahoo'', ''Sougo'' and ''Bing'' as the component search engines. However, MetSearch does not be built based on DHMAS, it does not have the self-adaptive mechanism.

To verify the performance of the proposed model, Success Rate (defined as (12)) and Mean Response Time (defined as (13)) are calculated for MetSearach and IM Search, respectively. The experimental results are shown in FIGURES 19 and 20.

$$SR = \frac{U_S}{U_T} \tag{12}$$

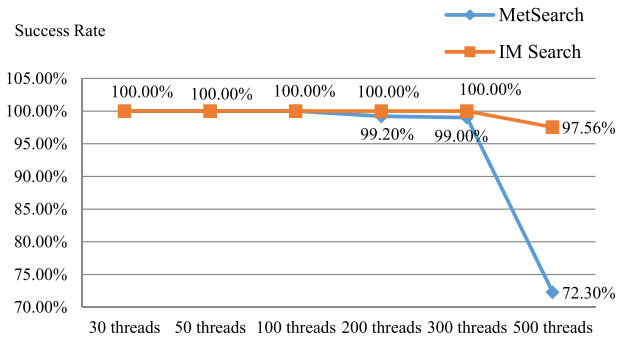

FIGURE 19. Experimental results in terms of Success Rate.

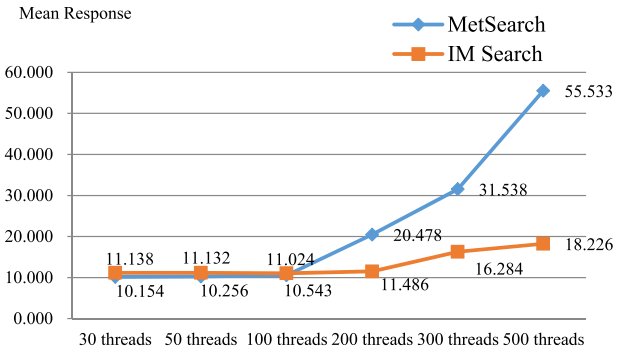

FIGURE 20. Experimental results in terms of mean response time.

where SR is the Success Rate, U_S is the number of users whose submitted queries are correctly responded, and U_T is the total number of users who submitted queries to meta-search engine.

$$MRT = \frac{\sum_{i=1}^{N} (TR_i - TS_i)}{N} \tag{13}$$

where MRT is the Mean Response Time, N is the total number of users who submitted queries, TS_i is the time when user i submitted the query request, and TR_i is the time when user i received the resulted results for the submitted query.

FIGURE 19 shows that, in the case of high concurrency, IM Search performs better than MetSearch in terms of Success Rate. In the ''medium load'' state, MetSearch has already experienced a decrease in Success Rate, while IM Search remains 100%. In the ''high load'' state, MetSearch has been reduced to 72.30%, however, IM Search still reaches 97.56%.

FIGURE 20 shows that, in terms of Mean Response Time, MetSearch performs slightly better than IM Search in the ''low load'' state. The reason is that the task assignment of MetSearch is simpler and more direct. While, in the case of high concurrency, IM Search has a better Mean Response Time, because it has the ability to adjust its structure based on the current load states.

Therefore, the experimental results indicate that IM Search is significantly better than MetSearch when dealing with

more complex tasks, that is, our proposed DHMAS is more effective than the component-based method to model the complex software systems.

V. CONCLUSION

In this article, we have presented an organizational structure and dynamic self-adaptive mechanism for a Holonic Multi-Agent System from a task-based perspective. Our proposed structure of a holon consists of a Decision Module, an Execution Module and a Task Queue. The Decision Module has the global information of the holon, which makes easier to manage the Holon. The Execution Module performs the specific tasks to meet system goals. Based on the proposed organizational structure, the self-adaptive mechanism including competitiveness adjust mechanism and structure adjust mechanism are also discussed, which guarantee the HMAS to adapt to the continuously changing environments. The self-adaptive adjustment mechanism will adjust the system structure based on the current competitiveness of each execution unit when the task environment changes, so that the system can perform task successfully. This improves the resource utilization. Moreover, two methods based on competitiveness are presented for task assignment. The heuristic search method based on competitiveness is proposed for the situation of a non-atomic parent holon assigning tasks for its sub-execution units, while the competitiveness -based contract net method is used for the situation of an atomic holon assigning tasks for Function Agents. The Task Assignment Module assigns tasks according to the current competitiveness of each execution unit, to ensure successful task execution. Our proposed model is very suitable for solving environmental anomalies when a small number of execution units fail. DHMAS can promptly schedule the competitive execution units to replace or assist the High-load or overload execution units to perform tasks, ensuring the system meets the environmental requirements. Finally, a metasearch engine based on the proposed model is implemented, and it has been proven that the proposed model has the ability to adapt to the changing environment.

However, there are still open issues that must be addressed:

1. improving the speed of the task assignment mechanism is a problem that must be solved;

2. identifying whether a Holon is in the state of High-load or not, is still a question to be explored, which will affect the performance of the self-adaptive mechanism.

REFERENCES

[1] X. Mao, Y. Qi, H. Wang, and B. Chu, "Agent oriented software development method," *Comput. Sci.*, vol. 30, no. 5, pp. 94–96, 2003.

[2] Z. Xiao, Z. Wang, and L. Xu, "A study of Jini-based mulit-agent systems," *Comput. Sci.*, vol. 29, no. 7, pp. 101–103, 2002.

[3] B.-Y. Wang and J.-H. Lü, "Software networks nodes impact analysis of complex software systems," *J. Softw.*, vol. 24, no. 12, pp. 2814–2829, Jan. 2014.

[4] L. G. Manescu *et al.*, "Complex software system for data management and analysis of power distribution grids," in *Proc. 4th Int. Conf. Math. Comput. Sci. Ind. (MCSI)*. New York, NY, USA: IEEE Computer Society, 2017, pp. 51–56.

[5] L. G. Manescu, D. Rusinaru, C. Popirlan, G. Stoian, M. Ciontu, G. C. Buzatu, M. Alba, and A. Cojoaca, "Complex software system for data management and analysis of power distribution grids," in *Proc. 4th Int. Conf. Math. Comput. Sci. Ind. (MCSI)*, Corfu, Greece, Aug. 2017, pp. 51–56.

[6] A. Bovenzi, F. Brancati, S. Russo, and A. Bondavalli, "An OS-level framework for anomaly detection in complex software systems," *IEEE Trans. Dependable Secure Comput.*, vol. 12, no. 3, pp. 366–372, May 2015.

[7] M. Salehie and L. Tahvildari, "Self-adaptive software: Landscape and research challenges," *ACM Trans. Auton. Adapt. Syst.*, vol. 4, no. 2, pp. 1–42, 2013.

[8] T. Vogel and H. Giese, "Model-driven engineering of self-adaptive software with EUREMA," *ACM Trans. Auton. Adapt. Syst.*, vol. 8, no. 4, pp. 1–33, Jan. 2014.

[9] H. Bruyninckx, M. Klotzbücher, N. Hochgeschwender, G. Kraetzschmar, L. Gherardi, and D. Brugali, "The BRICS component model: A model-based development paradigm for complex robotics software systems," in *Proc. 28th Symp. Appl. Comput.*, Coimbra, Portugal, Mar. 2013, pp. 1758–1764.

[10] H. Wang and B. Ding, "Growing construction and adaptive evolution of complex software systems," *Sci. China Inf. Sci.*, vol. 59, no. 5, pp. 1–3, May 2016.

[11] J. Brar and H. van der Meij, "Complex software training: Harnessing and optimizing video instruction," *Comput. Hum. Behav.*, vol. 70, pp. 475–485, May 2017.

[12] E. Adam and D. Trentesaux, "Dynamic self-organization in holonic multi-agent manufacturing systems," *Comput. Ind.*, vol. 66, pp. 99–111, Jan. 2015.

[13] R. Beheshti and N. Mozayani, "HOMAN, a learning based negotiation method for holonic multi-agent systems," *J. Intell. Fuzzy Syst.*, vol. 26, no. 2, pp. 655–666, 2014.

[14] S. Rodriguez, V. Hilaire, N. Gaud, S. Galland, and A. Koukam, "Holonic multi-agent systems," *Natural Comput.*, vol. 37, no. 3, pp. 238–263, 2011.

[15] M. Cossentino, N. Gaud, S. Galland, V. Hilaire, and A. Koukam, "A holonic metamodel for agent-oriented analysis and design," in *Proc. Int. Conf. Ind. Appl. Holonic Multi-Agent Syst. Holonic Multi-Agent Syst. Manuf. Regensburg*. Berlin, Germany: Springer-Verlag, Sep. 2007, pp. 237–246.

[16] M. Cossentino, V. Hilaire, N. Gaud, S. Galland, and A. Koukam, "The ASPECS process," in *Handbook Agent-Oriented Design Processes*. Berlin, Germany: Springer, 2014, pp. 65–114.

[17] B. Horling and V. Lesser, "A survey of multi-agent organizational paradigms," *Knowl. Eng. Rev.*, vol. 19, no. 4, pp. 281–316, Dec. 2004.

[18] H.-J. Bürckert, K. Fischer, and G. Vierke, "Transportation scheduling with holonic MAS-the TELETRUCK approach," in *Proc. Int. Conf. Practical Appl. Intell. Agents Multi-Agent Technol.*, London, U.K., 1998, pp. 577–590.

[19] A. Benaskeur and H. Irandoust, "Holonic approach for control and coordination of distributed sensors," Defence Res. Develop., Ottawa, ON, Canada, DRDC Valcartier Tech. Rep. 2008-015, 2008.

[20] F. J. M. Marcellino and J. S. Sichman, "A holonic multi-agent model for oil industry supply chain management," *Proc. Ibero-Amer. Conf. Adv. Artif. Intell.*, Lisbon, Portugal. Berlin, Germany: Springer-Verlag, Oct. 2010, pp. 244–253.

[21] M. Abdoos, N. Mozayani, and A. L. C. Bazzan, "Holonic multi-agent system for traffic signals control," *Eng. Appl. Artif. Intell.*, vol. 26, nos. 5–6, pp. 1575–1587, May 2013.

[22] S. Shafaei and N. G. Aghaee, "Biological network simulation using holonic multiagent systems," in *Proc. 10th Int. Conf. Comput. Modeling Simulation*, Cambridge, U.K., Apr. 2008 pp. 617-622.

[23] E. Adam, T. Berger, Y. Sallez, and D. Trentesaux, "Role-based manufacturing control in a holonic multi-agent system," *Int. J. Prod. Res.*, vol. 49, no. 5, pp. 1455–1468, Mar. 2011.

[24] M. Khazaei and N. Mozayani, "Overload management with regard to fairness in session initiation protocol networks by holonic multiagent systems," *Int. J. Netw. Manage.*, vol. 27, no. 3, p. e1969, May 2017.

[25] A. Pahwa, S. A. DeLoach, B. Natarajan, S. Das, A. R. Malekpour, S. M. S. Alam, and D. M. Case, "Goal-based holonic multiagent system for operation of power distribution systems," *IEEE Trans. Smart Grid*, vol. 6, no. 5, pp. 2510–2518, Sep. 2015.

[26] S. Rodriguez, N. Gaud, V. Hilaire, S. Galland, and A. Koukam, "An analysis and design concept for self-organization in holonic multi-agent systems," in *Proc. Int. Conf. Eng. Self-Organising Syst.* Berlin, Germany: Springer, 2006, pp. 15–27.

[27] A. Esmaeili, N. Mozayani, M. R. J. Motlagh, and E. T. Matson, "A socially-based distributed self-organizing algorithm for holonic multi-agent systems: Case study in a task environment," *Cognit. Syst. Res.*, vol. 43, pp. 21–44, Jun. 2017.
[28] J. Barbosa, P. Leitão, E. Adam, and D. Trentesaux, "Dynamic self-organization in holonic multi-agent manufacturing systems: The ADACOR evolution," *Comput. Ind.*, vol. 66, pp. 99–111, Jan. 2015.
[29] C. Călin, and C. Filote, "Artificial social models for holonic systems," *Holonic and Multi-Agent Systems for Manufacturing*. Berlin, Germany: Springer, 2011, pp. 133–142.
[30] M. Ulieru, D. Stefanoiu, and D. Norrie, "Holonic self-organization of multi-agent systems by fuzzy modeling with application to intelligent manufacturing," in *Proc. IEEE Int. Conf. Syst., Man, Cybern.*, Oct. 2000, pp. 1661–1666.
[31] R. Davis and R. G. Smith, "Negotiation as a metaphor for distributed problem solving," *Artif. Intell.*, vol. 20, no. 1, pp. 63–109, Jan. 1983.
[32] M. Wang, Q. Li, Y. Lin, and B. Zhou, "A personalized result merging method for metasearch engine," in *Proc. Int. Conf. Softw. Comput. Appl.* Bangkok, Thailand, Feb. 2017, pp. 203–207.
[33] Y. Li, Q. Li, and Y. Lin, "A personalized result recommendation method based on communities," in *Proc. Int. Conf. Data Mining, Commun. Inf. Technol.*, Phuket, Thailand, May 2017, pp. 1–5.
[34] J. Liu, Q. Li, and Y. Lin, "The classification of search results in the meta-search engine," in *Proc. Int. Conf. Comput. Sci. Netw. Technol.*, 2016, pp. 520–525.
[35] Q. Li, J. Wang, H. Chu, and T. Ji, "Personalization mechanism in an intelligent agent-based meta-search engine," *Scientia Sinica*, vol. 45, no. 5, pp. 605–622, 2015.

MEIJIA WANG received the M.E. degree from the College of Information Engineering, Northwest A&F University. She is currently pursuing the Ph.D. degree with Xidian University. Her main research interests include agent-oriented software engineering, data analysis, and social network analysis.

QINGSHAN LI received the Ph.D. degree from Xidian University. He is currently a Professor and a Ph.D. Supervisor with the School of Computer Science and Technology, Xidian University. His main research interests include agent-oriented software engineering, self-adaptive systems, and data analysis.

YISHUAI LIN received the Ph.D. degree from the Université de Technologie de Belfort-Montbéliard. She is currently a Lecturer with the School of Computer Science and Technology, Xidian University. Her main research interests include agent-oriented software engineering, knowledge management, and product design.