An agent-based business automated system with self-adjusting visibility for reliability

Hanh Pham\textsuperscript{a,*}, Yiming Ye\textsuperscript{b}

\textsuperscript{a}Department of Computer Science, State University of New York at New Paltz, 75 S Manheim Blvd, Suite 6, New Platz, NY 12561, USA
\textsuperscript{b}IBM T.J. Watson Research Center, P.O. Box 704, Yorktown Heights, NY 10598, USA

Received 30 October 2002; received in revised form 20 February 2003; accepted 28 February 2003

Abstract

This paper describes an agent-based model, called HAM, for business automation in large, distributed, and real-time systems. With resource bounds and time constraints, reliability and efficiency are difficult to achieve. We propose a self-restraining and self-stimulating control of agent interactions to meet the deadlines and to prevent agents from overloading system resources. The agent visibility and invisibility concepts are introduced and used to regulate the scopes of agent interactions and communications when it is needed. A mechanism, called DYVIREM, is designed to adjust agent visibility dynamically according to the deadlines and the resource limits. Through simulation experiments we also analyze the effects of agent visibility on the performance and on the quality of service of the proposed agent-based business model.

\textcopyright 2003 Elsevier B.V. All rights reserved.

Keywords: Business automation; Software agents; Visibility; Multi-agent systems; Control; Interaction; Real-time; Reliability; Resource bounds; Time constraints

1. Introduction

How should we build business automated systems?
Can we build them automatically? If we follow traditional ways, building a large number of different programs for a large number of transactions and data in different formats, it could be very costly and time consuming with additional handling of synchronization and compatibility. Software agents, which are autonomous programs having ability to act independently and intelligently, can be a good means in dealing with these distributed and synchronization features.

Agent technology has shown its potential in handling trades [9,35,45], auctions [35,48], data retrieval [3,24,27], supply chains and management [8,21,22,23,50], Internet search and usage [6,44,47],...
document processing [35,49] in an automatic and autonomous way. However, building large-scale agent-based systems for business automation is still a challenge as it involves a huge number of agents deriving from large numbers of business transactions and heterogeneous data. Since the computing resources, including CPU time, memory, database access, are bounded, agents are computationally limited [17,33,45,53]. Besides, the capacities of system resources are various at different times. How to meet the deadlines without overloading the system resources and how to use the available resources efficiently are important issues in building a reliable and effective agent-based business system.

In this paper we propose a self-restraining and self-stimulating control of agent activities to achieve system reliability and efficiency via a dynamic regulation of agent interaction scopes. A concept of ‘visibility’ is added to agent characteristics to represent and measure the degrees of agent interests in interacting with each other according to the deadlines, the capability and availability of the system resources.

The rest of this paper is organized as follows. A business agent-based model, called HAM, is described in Section 2. Reliability, in the context of time constraints and resource bounds, is discussed in Section 3. We explain the concepts of agent visibility and invisibility in Section 4. A mechanism for regulating agent visibility dynamically, called DYVIREM, is presented in Section 5. The effects of changing agent visibility are studied in Section 6. Finally, advantages and limitations of the proposed method and further research are discussed in Section 7.

2. A real-time agent-based model for business automation

To study how software agents can be used in business automation and how agents control their interaction scopes in business contexts we build a simple distributed and real-time business model (SDRBM) and a hypothetic agent-based model (HAM) to implement it. These models can be described as the follows.

2.1. A simple distributed real-time e-business model (SDRBM)

We assume that a business world \( W_b \) consists of:

- a set of business parties or organizations \( B \)
- a set of business ontology \( o_B \)
- a set of business logistics \( g_B \)
- a set of components \( c_B \)
- a set of resources \( R \)
- a set of activities \( V \)

Suppose that this business world \( W_b(B(o_B, c_B, g_B, R, V) \) has \( M \) business parties or organizations \( B = \{B_1, B_2, \ldots, B_M\} \), where \( M \) can be a dynamic number. A company, a customer, or an intermediary can be considered as a business party. Each business party \( B_i \) has a specification or ontology \( o_{Bi} \) of the given organization which defines its structures, such as which business components from \( c_B \) are included and the relationship between them. Thus, a business party may include a number of business components: \( c_{Bi} = \{C_1, C_2, \ldots, C_M\} \), where \( C_j \) has its predefined business task \( S_{ij} \), \( i = 1 \ldots M, j = 1 \ldots M \), which has a deadline \( dS_{ij} \), \( i = 1 \ldots M, j = 1 \ldots M \). In order to accomplish the given task the business component needs to define a plan of actions. An action can be: (i) an internal processing or (ii) an interaction/communication with other components.

As the system is distributed, the set \( R \) of resources \( R = \{R_1, R_2, \ldots, R_U\} \), which may include documents, databases, business knowledge, and materials at different locations, is managed in a decentralized way. Each resource \( R_u \), \( u = 1 \ldots U \), has a characteristic \( c_{Ru} \) representing the capacity and the availability of the given resource.

The business activities \( V \) consists of two kinds of actions: operations \( oV \) and contracts \( cV \), \( V = oV \cup cV \). Operations are regular, basic, and everyday activities and are predefined by business logistics \( g_B \). Con-
tracts are non-regular activities which are formed dynamically as the business world \( W_b \) progresses.

The business components and resources from \( cB \) and \( R \) are located at various locations. Each component has a physical address \( pA \), and a logical address \( gA \). While the physical address reflects the physical position of a business component or a resource, the logical address reflects other aspects related to different business characteristics.

In the context of e-business, we assume that each business component or resource can be represented by a software module which provides all necessary information about the current status, availability, etc., of the given business component or resource. Meanwhile, business ontology, logistics, and activities can also be represented and integrated into the system via corresponding software modules.

2.2. An agent-based approach for business automation in SDRBM

In building autonomous or partly autonomous business systems there are two main approaches: (i) traditional approach [19], where the system is built in a whole and its components are strongly related and depend on each other; and (ii) autonomous approach [21], where each component can be built independently and act on its own.

The first approach (i) is effective for one-time development of specialized systems. However, it is rigid, not easily to coordinate the components, and difficult to resize. If different components are built using different technologies and standards we would need to have broker systems, such as CORBA or DCOM, to transform intermediary data from one standard to the others. Unfortunately, such techniques are usually complicated. Therefore, it would be time-consuming and costly to develop and maintain business systems using them. Besides, the components in this kind of design still depend on each other and work together based on fixed coordination. In this case, the automation of the business system is achieved based on a carefully designed architecture of the entire system and of each component. If the business logistics are changed we may need to rebuild the existing systems. This approach can be used for building fixed structures but would not be suitable for building a general architecture for business automation.

The second approach (ii) can be realized using agent technology. The components of business systems can be built as autonomous software programs which implement functions of those components in an intelligent and flexible way. The components represented by such programs can act as autonomous agents: independent and able to change or fix itself. The interaction and data exchange between the components are carried out through agent interaction and communication. The agent approach can provide more general-purpose procedures with flexible cooperation and is more fault-tolerant. Business logistics can be embedded as agent knowledge and therefore when the business logistics are changed we do not need to rebuild anything, even the components. When an agent fails it is easy to replace and it can be done automatically, without stopping the system.

Therefore, we adopt the agent approach, for its autonomy and flexibility. The agent-based system for the business world \( W_b \) in the SDRBM now is an agent world \( W_a \) of \( N \) agents. There are two ways to build agent-based systems for business automation: (i) non-structured and (ii) structured. A non-structured agent system consists of universal and heterogeneous agents each of them represents a business party \( B^* \) and carries out the business automation steps: retrieving data, matching demands and supplies, executing transactions, etc., all together at once. The structured approach uses agents specialized in different roles which work together to carry out business automation step by step.

If we use the agent-based non-structured approach we would need to build different, complex, and customized agents for different business organizations. It would be very difficult to produce these agents automatically.

In this paper we choose the agent-based structured approach. The activities of business organizations are considered as a set of transactions [1] and are divided into seven basic categories or processing steps. Each processing step is carried out by specialized agents of a given type. The flows of business activities are maintained as regular workflow management [26,36,39–41,51], also by agents. In our model, to make the agent creation process automated
we build a bank of agent patterns BAP from which agents can be cloned or produced with modifications. The set BAP consists of several subsets of pattern agents to deal with the following issues:

1. **Interface Agents (IA)** for Interface automation of intelligent interactions with users.
2. **Data Agents (DA)** for Data automation of data retrieval, updating, and integrating.
3. **Linker Agents (LA)** for Analysis automation of data analysis and linking.
5. **Transaction Agent (TA)** for Transaction Automation of transaction forming.

Thus, we have $\text{BAP} = \{\text{IA, DA, LA, RA, TA, EA, WA}\}$ where each of agent subsets may have a number of pattern agents for the given category as shown in Fig. 1.

The input and output of this heterogeneous agent-based model HAM for business automation in SDRBM are the data exchanged with the human users and with other systems which could be databases, the Internet, or equipments such as robot sensors, PDAs, video cameras, scanners at the cashiers, counters in the warehouses, etc., as shown in Fig. 1. There are two types of pattern agents to carry out the input and output of HAM: the interface agents (IA) and the data agents (DA). The other types of agent patterns deal with internal processing inside the agent world Wa.

### 2.2.1. Interface agents

Interface agents, IA, are built for a user-friendly, customized, and intelligent interface of the HAM
with human users. If there are \( L \) types of such interactions with users then we would need to have \( L \) patterns of interface agents: \( \text{IA} = \{\text{IA}_1, \text{IA}_2, \ldots, \text{IA}_L\} \). However, in an automated business system the request agents may be given in different formats and hidden among other data such as emails, files, etc. Therefore, the request agents \( \text{RA} \) are needed in SDRBS to generate business requests even when they are implicitly represented. Thus, SDRBS would have a number of request agents from \( \text{RA} = \{\text{RA}_1, \text{RA}_2, \ldots, \text{RA}_G\} \), where \( G \) is the number of patterns agents of this category or number of the business request types. As the system is a real-time one, each request has a deadline, either hard or soft. The deadline for each request is defined automatically based on the user input from the Interface Agents or the retrieved data from the Data Agents.

### 2.2.2. Data agents

In HAM, business data can be distributed and heterogeneous coming from: (i) data sources such as databases, files, the Internet, robot sensors; or (ii) interactions with people such as emails, on-line dialogues, phones, faxes. The data agents \( \text{DA} \) need to retrieve business data in different formats and then convert them into a common agent format (CAF). Thus, each kind of data format or interaction would need a pattern agent. \( \text{DA} = \{\text{DA}_1, \text{DA}_2, \ldots, \text{DA}_K\} \), where \( K \) is the number of patterns agents of this category.

### 2.2.3. Linker agents

Since the business data are usually related to each other, such as purchased items and the customer profile, or need to be verified, such as credit card numbers, the linker agents are needed to analyze and link the data automatically with each other. Besides, as the modern business system is usually overwhelming with huge amount of data, mining and warehousing available data should also be done and can be carried out by the linker agents \( \text{LA} = \{\text{LA}_1, \text{LA}_2, \ldots, \text{LA}_H\} \), where \( H \) is the number of patterns agents of this category.

### 2.2.4. Request agents

In many e-business systems it is assumed that the requests, either demands or supplies, are available immediately as the customers make the requests explicitly, for example fill in forms in web pages. However, in an automated business system the requests may be given in different formats and hidden among other data such as emails, files, etc. Therefore, the request agents \( \text{RA} \) are needed in SDRBS to generate business requests even when they are implicitly represented. Thus, SDRBS would have a number of request agents from \( \text{RA} = \{\text{RA}_1, \text{RA}_2, \ldots, \text{RA}_G\} \), where \( G \) is the number of patterns agents of this category or number of the business request types. As the system is a real-time one, each request has a deadline, either hard or soft. The deadline for each request is defined automatically based on the user input from the Interface Agents or the retrieved data from the Data Agents.

### 2.2.5. Transaction agents

These agents are needed to establish the transactions automatically. They need to match business supplies to business demands and to form the transactions implementing a given business logistics or contract. Since matching of business requests may be substantially different from one transaction or logistics to the others, for each type of them we may need a pattern agent from \( \text{TA} = \{\text{TA}_1, \text{TA}_2, \ldots, \text{TA}_Q\} \), where \( Q \) is the number of patterns agents of this category. The transaction deadlines are defined by the request deadlines which are predetermined by the Request Agents.

### 2.2.6. Execution agents

To implement the transactions of business logistics or contracts we need execution agents from the pattern agents \( \text{EA} = \{\text{EA}_1, \text{EA}_2, \ldots, \text{EA}_P\} \). These agents carry out separate transactions based on the descriptions of business logistics or contracts and based on the rules given in the business knowledge bases and business databases. As a transaction may include or trigger the other transactions, an Execution Agent may then create a Request Agent for the next transaction formation.

### 2.2.7. Workflow agents

Implementation of related transactions [2] is more sophisticated as it requires coordination and conflicts solving. We need workflow agents \( \text{WA} \) to make the systems capable of accepting new logistics, updating and integrating it with the others automatically, and guarantee the cyclicity and the flexibility in the
implementation of the transactions. Depending on the number of coordination and conflict types we may have \( WA = \{ WA_1, WA_2, \ldots, WA_F \} \), where \( F \) is the number of pattern agents of this category.

Thus, the population of agents for business automation in SDRBM can be created from the set of agent patterns \( \text{BAP} = \{ IA, DA, LA, RA, TA, EA, WA \} \). The general architecture of HAM in regards to the business world \( Wb(B(oB, cB, gB), R, V) \) from SDRBM is shown in Fig. 2.

### 2.3. Mapping of the agent world \( Wa \) into the business world \( Wb \)

How agents in \( Wa \) are created and represent the business world \( Wb(B(oB, cB, gB), R, V) \)?

1. The agent world starts with an interface agent \( IA_0 \), a data agent \( DA_0 \), and a workflow agent \( WA_0 \). In the beginning \( B = \emptyset \).
2. A business party or organization \( B^* \) is created and needs to be represented by agents in \( Wa \).
3. Agents \( IA_0 \) and \( DA_0 \) input data about the business ontology \( oB^* \), components \( cB^* \), and logistics \( gB^* \) for the given organization \( B^* \).
4. Agent \( WA_0 \) adds \( B^*(oB^*, cB^*, gB^*) \) into the set of organizations \( B \).
5. New agents \( DA \) and \( WA \) dedicated for \( B^* \) are created, \( DA \) for holding/representing \( oB^*, gB^* \) and \( WA \) for processing them.
6. The workflow agents \( WA \) of \( B^* \) create new linker agents \( LA \) of \( B^* \) based on the given business ontology \( oB^* \) (linker agents \( LA \) carry out the jobs of various business components \( cB_i \in B^* \) in the business world \( Wb \)).
7. Agents \( WA \) create new request agents \( RA \) and maybe transaction agents \( TA \) (*) based on \( gB^* \).

\(^1\)Notice that business transactions include the regular operations \( oV \) and as well as the activities based on the contracts \( eV \) from the business world \( Wb \). If a transaction is an operation from \( oV \), i.e., a regular activity, the corresponding transaction agent \( TA \) is created by the workflow agents \( WA \). If a transaction derives from a contract from \( eV \), i.e., a non-regular activity, the corresponding transaction agent \( TA \) is created when all the needed request agents participating in the given contract are matched.

---

**Fig. 2. General structure of agent-based business system by HAM.**
(8) Linker agents LAs collect, analyze, and link necessary data from different DAs and submit to the request agents RAs based on $gB^*$. 
(9) Request agents RA form the demands and supplies. 
(10) Agents TAs and RAs match the demands and supplies from the current requests represented by RAs and form the transactions. After that agents TAs register the transactions with agents WAs. 
(11) The workflow agents WAs create the corresponding execution agents EAs for the given transactions. 
(12) Agents TAs submit logistics of transactions to agents EAs. 
(13) The agents EAs implement transactions. During or after the implementation EA may submit/ change data in DAs which may lead to the creation of new demands, supplies, or transactions. 
(14) When a transaction is finished its hosting agent EA is terminated. 
(15) Data about the business resources $R$: customer orders, manufacturing, and inventory from outside are entered through the data agents DA or interface agents IA and can participate immediately into related business processes: forming demands, supplies, triggering transactions, etc. 
(16) If the business $B^*$ in $Wb$ is removed then all its related agents (IA, DA, LA, RA, TA, EA, WA) are deleted from $Wa$. 

3. Achieving reliability by regulating agent interaction scopes

While the use of software agents can make the development of business components less costly and more automatic agent-based systems would require more computation power and resources than the regular models. Therefore, reliability, which is already critical in business systems, becomes an even more important issue in agent-based business systems where there are also time constraints and resource bounds. 

In large agent-based business systems, large agent populations may run on computers which have limited computing resources. It is possible that some agents can be starved of resources and cannot accomplish their tasks before the deadlines. In the worst case, the agents may overload the system and crash the hosts. Therefore, the question here is how to efficiently use or distribute the limited computing power to these agents. 

Almost all agent activities consume resources, however, to control and regulate all of agent activities would be extremely complicated and would have significant effects on agent performance. As interactions including communication between agents contribute a big portion to the agent’s overall resource consuming we will focus on regulating agent interactions as a key way to achieve the over all system reliability. 

In the described agent-based model HAM for business automation, while there is a number of various interactions of different kinds of agents, as shown in Fig. 2, the heaviest volume lies in the interactions among the request agents RAs, the transaction agents TAs, and between them in matching the requests to form the transactions. In fact, HAM may include many e-marketplaces. Recall that a business organization $B^*$ in HAM is vertically constructed based on the predefined structure of seven automation steps as described in Section 2.2. The agents, which carry out those seven steps of the predefined structure, are automatically created from the set of agent patterns $BAP = \{IA, DA, LA, RA, TA, EA, WA\}$. Thanks to this predefined automation structure the agent communications and interactions within an organization $B^*$, i.e., vertically, are predefined and therefore we can avoid the their optimization [54]. However, horizontally, the interactions among the agents RAs and TAs which belong to different organizations $B_1, B_2, \ldots, B_M$, are not predefined as no relationship between the organizations is given. A transaction agent TA or a request agent RA may need to do an exhausted search among all other agents TAs and RAs to find the needed partners. 

If the business system is large and distributed, the number of agents and consequently the communication overhead and search time in these e-marketplaces could be very large. If each RA interacts with all other RAs in the system, then the time that an agent needs to select its best match could be much more than the time constraint of the request repre-
sented by the given agent. For example, a system which has $N_{ra} = 100 000$ of RAs agents would have to handle $10^{98}(10^8 - 1)/2$ or more than $10^{11}$ interactions in total, or 99 000 per agent! The problem here is not only about the time but also about the resources such as memory or CPU time which are needed to support such a large scope of agent interactions.

In order to keep the systems reliable we need to distribute the available resources to the agents fairly and dynamically, according to the agent deadlines and to the system capacity. However, in large systems it is difficult to impose a centralized system control on every component in terms of resource usage and time constraints [18, 24, 25, 38].

In distributed systems, to protect the local hosts, there are techniques such as sand box for Java-based programs [31, 34, 37] or resource and execution restrictions for web browser-based programs [5, 10]. However, those approaches may not be efficient and suitable for agents, because: (i) there is only a small number of browse-based and Java-based programs running at a time while the number of agents at a host could be very large and unpredictable; (ii) The range of the resources needed to support an agent is much wider than the range of what a web-based or a browser-based program would need. The amounts of resources consumed by agents are hard to predetermine as they could be changed dynamically depending on the activities of the agents and as the number the other agents can be changed as well.

In agent-based systems, resource limits and time constraints have been considered in recent works [6, 7, 33, 42, 45]. Most of proposed methods are designed for specific scenarios such as bargaining [33, 45], Internet search [6], visualization [46], or at abstract and cognitive levels [7, 15].

In the case of HAM, a general-purpose agent-based business model, to achieve the stability and to meet the business deadlines it is essential to control and efficiently adjust the number of interactions or the scope of communication based on the currently available resources and the time constraints. Our goal is to integrate a self and adaptive control mechanism into the structures of agents in BAP.

Changing search scopes has been a subject of research for a long time [11, 30]. Abstraction [16, 28], localizing [12, 32], best-first [29], branch and bounds [14], and nearest neighbor search [52, 56] are some of different ways to reduce the search space when the size of the full search space is too large.

In this paper we develop an adaptive approach in reducing the search space for agents which look for their suitable partners to form transactions. Our goal is to build a space reduction mechanism which is flexible enough so that it can be applied in different contexts of interactions as described in Section 2. Meanwhile, this mechanism should not be too complicated to generate high overheads. In order to reach those goals we will develop a measurement unit to determine the needs of agent interactions and the capacity of the system in supporting it. The next section describes that measurement, which is the foundation for building the search space regulation mechanism described in Section 5.

4. Agent visibility and invisibility

In order to represent the agent ability and its interest in recognizing and interacting with other agents we develop a concept called agent visibility.

4.1. Basic concepts

Agent visibility is the ability and interest of an agent, an autonomous program, to be aware of the existence of other agent programs and their features. Invisibility is its opposite, when the agent does not recognize the other agents either because it does not have that ability or that function is not desired and is turned off.

This feature of agents is an imitation of a similar behavior of humans. In the real world, if a person goes shopping at a very big mall with limited time and strength she would not try to look at every single store and at every single item in the mall but rather focus on a few of stores and items there. As one of the main goals in designing agents is to make them compatible and behave similarly to people [13], we incorporate this characteristic into agents and use agent invisibility to reduce the scope of agent interactions in our agent-based business model HAM.

To attain this goal we need to: (i) define the attributes for agent visibility; (ii) calculate the vis-
4.1.3.3. VIR-3 (Non-transition)

If Ap and Ar can interact with each other and Ar and Aq can interact with each other, then there is no guarantee that Ap and Aq can interact with each other. It is easy to find examples of VD(Ap, Aq) ≤ μ^s = R^s while VD(Ap, Ar) ≥ μ^s, VD(Ar, Aq) ≥ μ^1 = R^1, VD(Ap, Ar) ≥ μ^3. This feature makes the encounters of agents unpredictable and also makes rooms for agents to become brokers, when {Ap→Aq} and {Aq→Ap}, i.e., Ap and Aq cannot see each other but {Ar→Aq}, i.e., Ar can see both of them. Then, Ar can become the broker for Ap and Aq to exchange data or to make a business transaction (Fig. 3).

4.2. Agent visibility with time and resource constraints

When a customer order or a business task has a deadline, the agent actions and requests have time constraints and they should be considered in regulating the agent interaction degree via setting the visibility of agents. Related works on real-time agent-based systems [4] mostly do it at the agent level using self-control mechanisms. For example, the agent keeps interacting with other agents and chooses whatever is best from the currently known options, until the deadline is reached.

When the system resources such as memory or CPU time are limited, we have resource bounds or constraints. Then, the agent interaction should also be limited accordingly to those constraints. Related works on limiting agent activities [4,6,7,15,33,45] can be divided into two categories: (i) global, by setting general limits, for example, whenever the number of interactions of the whole system reaches a limit the system does not allow more interactions; (ii) local, by setting limits for each agent on the usage of memory, CPU time, for example, using a sandbox for each agent.

4.2.1. Handling the time constraints

We regulate the agent interaction by taking the deadlines into account right from the beginning, before the deadlines are reached. Assume that the next deadline of agent Ai is reflected in its attribute TLi which shows the time left until the deadline. We add TLi and highlight it in the formula (1) so that the time constraint TLi plays an essential role in making the given agent visible to the others. In other words, agents with nearer deadlines may look clearer than the agents with later deadlines. This can be done as follows:

\[
VD(A_i, A_j) = \frac{\text{sign}(\text{TL}_i + \Delta_i) \times \text{sign}(\text{TL}_j + \Delta_j) \times PI}{\text{TL}_i + \Delta_i + \Omega_i} \times \left( \prod_{k=1}^{K} \left[ \text{sign}(\text{V}_k^i) \times \text{sign}(\text{V}_k^j) \times \text{ED}_k^i \right] \right) \]  

where, \(\Delta_i\) and \(\Delta_j\) are the late tolerances, respectively, for \(A_i\) and \(A_j\).

• in soft real-time systems, where the deadlines are allowed to missed: \(\Delta_i, \Delta_j \geq 0\) and the agents with missed have the highest priorities.

• in hard real-time systems, where any result provided after the deadlines cannot be used: \(\Delta_i, \Delta_j = 0\) and the agents with missed deadlines should be dismissed.

\(\Omega_i\) and \(\Omega_j\) are the adjust amounts to lower the upper limit of VD domain. When TLi and TLj are very close to zero VD(Ai, Aj) could be a very large number. For example, with \(\Omega_i = 1\) and \(\Omega_j = 1\) and \(\Delta_i, \Delta_j \geq 0\) we have VD(Ai, Aj) ≤ 1.

4.2.2. Handling the resource constraints

Assume that the total CPU and memory utilizations which can be granted to N agents are \(G_{c_{\text{cpu}}}\) and \(G_{m}\). The currently used utilizations are \(C_{c_{\text{cpu}}}\) and \(C_{m}\). The reservations for incoming agents are \(R_{c_{\text{cpu}}}\) and \(R_{m}\). The maximum CPU and memory usage for an agent interaction are \(A_{c_{\text{cpu}}}\) and \(A_{m}\). Se is a security

![Fig. 3. Agent broker in non-transition visibility.](image-url)
quantity which makes the permitted number of interactions PI lower than the real limit. Thus, the projected number of interactions which is allowed per agent and in average, to be carried out without causing the system overload, can be defined as the following:

\[ PI = \frac{1}{N} \min \left( \left[ \frac{G_{CPU} - (C_{CPU} + R_{CPU})}{2A_{CPU}} \right], \left[ \frac{G_{m} - (C_{m} + R_{m})}{2A_{m}} \right] \right) - \frac{1}{N} S_e \]

(3)

This rough estimation can be used in regulating the visibility degree to increase or decrease the number of interactions depending on whether the available resources utilization is high or low as follows:

\[ VD(A_i, A_j) = \text{sign}(\ln(PI)) \times PI \]

\[ \times \left( \sum_{k=1}^{K} \left[ \text{sign}(\Psi(V_k^i)) \times \psi(V_k^i) \times ED_k^i \right] \right) \]

(4)

If PI < 1 then \( \text{sign}(\ln(PI)) = 0 \) and it would make VD(Ai, Aj) = 0 for all (Ai, Aj), i, j = 1...N. In other words, if the available resources are not enough to support further interactions, in this case: one per agent, the visibility of all agents would turn to 0 and therefore we can immediately prevent further agent interactions.

Even though the agents are different, i.e., agents may have different CPU and memory usage for interactions, the use of the maximums of CPU and memory usage among agent interactions in estimation and the use of the averaged-by-agent PI as the limit defined for all agents would guarantee that in total the number of interactions allowed will be safe for the system.

Since the agent visibility depends on the current usage of CPU and memory utilization, the visibility degrees are dynamic. When the occupied resources are released, the value of PI would be increased, increasing the visibility degrees of all agents in the system. As the number of agents N is dynamic, we used the security quantity Se and the reservations for incoming agents \( R_{cpu} \) and \( R_{m} \) to reserve the needed resources for those agents with an assumption that right in the beginning of their creations those agents will have not have any interaction. Once when the incoming agents are created and registered in the system they are officially counted in N and therefore their interaction forecasts will be taken into account in the new value of PI.

In order to consider both time and resource constraints we can combine (2) and (4) as the following:

\[ VD(A_i, A_j) = \frac{\text{sign}(\ln(PI)) \times \text{sign}(TL_i + \Delta_i) \times \text{sign}(TL_j + \Delta_j) \times PI \times (TL_i + \Delta_i + \Omega_i) \times (TL_j + \Delta_j + \Omega_j)}{(TL_i + \Delta_i + \Omega_i) \times (TL_j + \Delta_j + \Omega_j)} \]

\[ \times \left( \sum_{k=1}^{K} \left[ \text{sign}(\Psi(V_k^i)) \times \psi(V_k^i) \times ED_k^i \right] \right) \]

(5)

The domain of VD is \([0, \Lambda]\), where \( \Lambda \) is the maximal value of PI, i.e., the maximal number of interactions allowed per agent.

5. A mechanism for dynamically self-adjusting visibility

In order to manage the scope of agent interaction we build a mechanism called DYVIREM for defining agent visibility dynamically so that it can regulate the scope of interactions accordingly. It has the following steps:

Step-1: As agent Ai is registered in the system, it defines following data:

- \( \text{EVAi} = \{V_{i1}, V_{i2}, \ldots, V_{iK}\} \)
- \( \{ED_{i1}, ED_{i2}, \ldots, ED_{iK}\} \)
- threshold \( \mu_i \)
- time left until the deadline \( T Li \)

Instead of submitting all of these ‘raw’ data, agent Ai calculates and submits the followings to the Visibility Service (VS):

- \( \psi(V_k^i), ED_k^i, k = 1 \ldots K \)
- \( \mu_i, (TL_i + \Delta_i) \) and \( \text{sign}(TL_i + \Delta_i) \)

Notice that \( \psi(V_k^i), k = 1 \ldots K \), need to be calcu-
lated only once and are updated later only if there is any change. In any case, the calculations of \( VD(A_i, A_j), i, j = 1 \ldots N \) are distributed between agents and the system Visibility Service.

**Step-2:** If \( \text{sign}(TL_i + \Delta_i) = 0 \) then agent \( A_i \) will not need to calculate anything, only to submit this value to the system Visibility Service. The VS maintains the visibility matrix as shown in Fig. 4, it is built in the similar way as in [42,43,55]. The VS sets or updates its visibility matrix with \( VD(A_i, A_j) = 0, \forall j = 1 \ldots N / j \).

**Step-3:** If \( \text{sign}(TL_i + \Delta_i) \neq 0 \) then the Visibility Service defines \( PI \) based on formula (3) and calculates the visibility degrees based on formula (5) and the data from \( A_i, A_j, i, j = 1 \ldots N \) and \( i \neq j \).

**Step-4:** When an agent \( A_j \) wants to search for business partners in general it can use the Visibility Service in the general mode. The VS returns the list of agents \( \{A_1, A_2, \ldots, A_i\} \) which can interact with the given one, i.e., so that: \( \{A_j \leftrightarrow A_i\}, \{A_i \leftrightarrow A_j\}, \ldots, \{A_i \leftrightarrow A_j\} \).

**Step-5:** In order to retrieve that list for agent \( A_i \) the VS looks at the row \( A_i \) and checks for each cell \( D(A_i, A_j), j = 1 \ldots N / i \) of the Visibility Matrix. For every value \( D(A_i, A_j) \) which is more than \( \mu_i \), the threshold of the given \( A_i \), the VS checks further if its symmetric value \( D(A_j, A_i) > \mu_i \). If so then according to VIR-3 the agent \( A_i \) and \( A_j \) are set for an interaction with each other.

Fig. 4 shows how the visibility degrees can change the scope of agent interaction. For agents \( A_2 \) and \( A_{11} \), there are virtual circles of radii \( R_i \) and \( R_j \), where \( R_i = \mu_i^2 \) and \( R_j = \mu_j^1 \). Only agents that fall inside the virtual circle is visible to the owner agent. When the agent visibility degrees are changed due to changes in time and resource constraints, the dis-

![Visibility Matrix](image)

**Fig. 4. Defining visibility in DYVIREM.**
tributions of its visible agents falling inside these circles would also change, leading to change of scope of the agent interaction.

5.1. Computation complexity

In the worst case and without parallel processing, the computation complexity of the described mechanism DYVIREM is $X \times K^2 \times N$ where $X$ is the time for each agent to calculate $\psi(V_i^k)$, $k = 1 \ldots K$, and $K$ is the number of effecting attributes, $N$ is the number of agents in the system. However, since agents can interact with the Visibility Service in parallel, $N$ may be reduced to $N/n$ where $n$ is the parallel degree of the given system. Besides, not all of the attributes will be changed all the time and when sign$(T_{Li} + \Delta i) = 0$ the Visibility Service can skip $K^2(x-1)$ in calculating $VD(A_i, A_j) = 0$, $\forall j = 1 \ldots N/j$, where $x$ is the number of $A_i$ which has sign$(T_{Li} + \Delta i) = 0$. In the best case when $PI < 1$ all of $VD(A_i, A_j) = 0$, $\forall i, j = 1 \ldots N$ is set without any calculation. Thus, the computation complexity for calculating agent visibility for all agents is $X \times K^2 \times (N - x) \times N/n$, where $X$, $K$, $x$, $n$ are either constants or not significantly changing while $N$ increases. However, as agents can act independently and in parallel with each other the computation complexity for each agent is much more important than the computation complexity for all agents.

Finally, formula (5) is implemented in the way so that, at any time, if a co-product in it is 0 then the rest of the calculation for the given $VD(A_i, A_j)$ will be cancelled as $VD(A_i, A_j) = 0$ immediately. Therefore, the actual computation time of $VD(A_i, A_j)$ would be much less than $N^2$ and would not become a significant overhead in the agent search time, as shown in the simulation results presented in the next section.

6. Simulation results: impacts of agent visibility

We have conducted simulation experiments to study the influences of agent visibility on the agent interactions and their search results. Large populations of agents are generated with dynamic visibility. Each agent represents a business unit and interacts with other agents. Agents search for the needed partners to interact, when interests are matched and other conditions are met the transactions are formed. Instead of letting all agents ‘see’ each other and interact with each other we use DYVIREM to regulate the visibility of all agents. We investigate how the reduction and the increase of agent visibility would have effects on: (i) the performances of agents, shown by the search time to form the transactions; and (ii) quality of service of agents, shown by the number of transactions.

Different visibility scales of the described agent-based business world are represented by a parameter called visibility density $D_{vis}$. It shows the percentage of agents which can ‘see’ each other in the agent-based business world: $D_{vis} = V/N/(N - 1)$, where $V$ is the number of visible pairs of agents which can ‘see’ each other and $N$ is the number of agents. Fig. 5 shows a number of agents which belong to a business organization and the visible interaction/communication links between them. Entering the system as independent agents for different tasks these agents automatically find the related partner agents through the interactions with the agents which are visible to them, defined by DYVIREM. The displayed links are established using dynamic visibility.

We change the capacity of the systems: available CPU and Memory, which leads to the changes of $PI$ the number of allowed interactions, so that the visibility density $D_{vis}$ is changed in the range of $[0,1]$ or $[0-100\%]$. We measure how the time $T_{sear}$ for searching the needed partners to form transactions and how the number of transactions $Q_{tran}$ would be changed. The results are shown in Table 1.

It shows that when the visibility level reduces, the number of transactions is also reduced. However, the searching time decreases more substantially. In large-scale systems if we keep 100% visibility, i.e., each object can ‘see’ all the others, then the searching time is very long and could be unacceptable. We can reduce the visibility until the searching time satisfies the time constraints. For examples, from the simulation results in Table 1, if we reduce the visibility to 0.689 or almost 70% then the number of transactions reduces to 83 or 17% less but the searching time would reduce to 69 or 31% less!
Further, if we reduce the visibility to 0.378 or 39% then the agents can still form 58% of all possible transactions for 39% of time or 2.5 times less compared with the case when $D_{vis} = 1$, i.e., without Visibility density Search time Number of transactions

<table>
<thead>
<tr>
<th>Visibility density ($D_{vis}$, [0,1])</th>
<th>Search time (Tsearch, %)</th>
<th>Number of transactions (Qtrans, %)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>0.011</td>
<td>1.211</td>
<td>0.000</td>
</tr>
<tr>
<td>0.156</td>
<td>16.556</td>
<td>25.000</td>
</tr>
<tr>
<td><strong>0.378</strong></td>
<td><strong>38.778</strong></td>
<td><strong>58.333</strong></td>
</tr>
<tr>
<td><strong>0.500</strong></td>
<td><strong>51.000</strong></td>
<td><strong>75.167</strong></td>
</tr>
<tr>
<td>0.600</td>
<td>61.320</td>
<td>75.352</td>
</tr>
<tr>
<td><strong>0.689</strong></td>
<td><strong>69.283</strong></td>
<td><strong>83.333</strong></td>
</tr>
<tr>
<td>0.778</td>
<td>78.778</td>
<td>83.978</td>
</tr>
<tr>
<td>0.844</td>
<td>85.544</td>
<td>84.102</td>
</tr>
<tr>
<td>0.867</td>
<td>86.867</td>
<td>84.834</td>
</tr>
<tr>
<td>0.889</td>
<td>89.981</td>
<td>85.373</td>
</tr>
<tr>
<td>0.911</td>
<td>92.131</td>
<td>86.398</td>
</tr>
<tr>
<td>0.933</td>
<td>94.313</td>
<td>86.943</td>
</tr>
<tr>
<td>0.956</td>
<td>96.556</td>
<td>87.223</td>
</tr>
<tr>
<td>0.967</td>
<td>97.667</td>
<td>88.657</td>
</tr>
<tr>
<td>0.978</td>
<td>98.778</td>
<td>90.135</td>
</tr>
<tr>
<td>0.989</td>
<td>99.889</td>
<td>91.667</td>
</tr>
<tr>
<td>1.000</td>
<td>100.000</td>
<td>100.000</td>
</tr>
</tbody>
</table>

Thus, by regulating the visibility of agents we can make the agent search more flexible and adaptive to the changes of the time requirements and to the system capacity. From the experiments we observed an important and useful result: when the density and the similarity of the business data are high, i.e., when we have a lot of business requests but many of them are similar, we can decrease the visibility of agents, thus decrease the resource consumption and the searching time without reducing the searching quality substantially.

7. Conclusions

We have proposed the agent-based model HAM
for carrying out business automation in large, distributed, and real-time business systems. In this model, the agent-based components of a business organization can be created and integrated automatically into the system. To achieve system reliability we focus on controlling the agent interactions. In order to regulate the scope of agent interaction we have proposed the agent visibility/invisibility concepts and the mechanism DYVIREM for dynamically regulating the agent visibility. We have described how time and resource constraints can be taken into accounts in defining the scope of agent interactions.

The experimental results show that the agent visibility concept can be a useful tool to regulate the amount of agent interactions allowing the agents to meet deadlines without overloading the available but limited resources.

However, more works need to be done in defining more simple and effective ways to calculate visibility with more complicated business contexts and logistics. For example, when an agent has several business interests which may not relate to each other, it will not be effective to combine all visibility-effecting attributes to calculate one overall visibility degree and use it for different interests. A concept of multi-dimensional visibility would be needed, where the visibility by each dimension for each business context is defined separately and is used independent from each other.

Research on establishing relationships and connections between visibility among agents such as group visibility and level visibility based on priorities may also help to reduce or avoid the visibility calculations. Visibility conditions for broker agents to have profits could also be an interesting topic to explore. Meanwhile, we will move the experiments from the simulation level to real agent platforms.

References


