Automated assembly of Internet-scale software systems involving autonomous agents

Wenpin Jiao\textsuperscript{a,b}, Yanchun Sun\textsuperscript{a,∗}, Hong Mei\textsuperscript{a}

\textsuperscript{a} School of Electronics Engineering and Computer Science, Peking University, China

\textsuperscript{b} Key Laboratory of High Confidence Software Technologies (Peking University), Ministry of Education, China

\textbf{A R T I C L E   I N F O}

Article history:
Received 14 October 2009
Received in revised form 22 March 2010
Accepted 30 April 2010
Available online 15 June 2010

Keywords:
Autonomous agent
Software assembly
Software quality
Internet-scale software systems

\textbf{A B S T R A C T}

On the Internet, there is a great amount of distributed software entities deployed independently and behaving autonomously. This paper describes an automated approach to constructing Internet-scale software systems based on autonomous software agents. In the approach, the systems are modeled by interconnected divisions and cooperative roles. The approach adopts a dynamic trial-and-evaluation strategy to select high quality autonomous agents to undertake the responsibilities of roles, and implements a special mobile agent, called delegate, carrying the interaction information specified for responsibilities of roles to facilitate the interoperations among autonomous agents. The experiments show that the approach is highly scalable and improves the overall qualities of systems remarkably.

© 2010 Elsevier Inc. All rights reserved.

\section{1. Introduction}

As the Internet has been transforming from the provider of content services to the platform of computation services, the mode of constructing software systems has changed dramatically. On the one hand, software entities are dynamically and independently deployed and located at different nodes on the Internet, they are decidedly autonomous and long-lived and they may not be invoked in the sense as traditional components (Garlan, 2002). They have many new characteristics (Garlan, 2002; Shaw, 1999), for example, they can autonomously enter or leave the Internet environment and decide whether to provide services or not. Usually, people understand autonomy of software entities as the taking of independent action. Software entities are autonomous implies that they interact without any intervention from users or other controlling entities (Suzuki and Suda, 2005) and take actions upon their own decisions. Internet-scale software systems referred to as Internetware in (Mei et al., 2006) have to be dynamically formed as loosely coupled software alliance (Shaw, 1999) by incorporating independent and autonomous software entities spanning different organizations (Wijngaards et al., 2002). In this paper, software entities with autonomy are referred to as autonomous agent (Abbr. as AG).

On the other hand, there are a large number of AGs offering the same or similar services with variety of QoS (quality of service) on the Internet. Developers have more opportunities for selecting services providers to construct their software systems according to their functional and non-functional requirements. Therefore, how to construct high quality internet-scale software systems via automatically selecting and assembling AGs has become one of the grand challenges in the era of the Internet.

For approaches to automated Internet-scale software assembly, the following properties are desired at least.

\begin{itemize}
  \item Convergence/guaranteed success: Software systems are required to meet the requirements of users and the assembly approach should be feasible to select appropriate AGs to form the systems that achieve the specified objectives.
  \item Distribution and local autonomy preservation: Running on a true distributed environment, Internet-scale software systems are naturally decentralized, i.e., there is not a centralized role controlling or coordinating the cooperation of involved AGs. In addition, when participating in the systems, AGs are not obliged to provide services and they should also be prevented from becoming the communication bottlenecks due to over-engagement.
  \item Optimizing system QoS: A large number of existing AGs on the Internet offer an opportunity for adaptively selecting appropriate candidates with desired QoS in the assembly of software systems.
  \item Simplicity and efficiency: While selecting and assembling AGs, the approach should support a quick and simple way for constructing stable software systems.
\end{itemize}
Currently, in the research areas such as multiagent systems and Web services, much work has coped with the automated integration of software systems via composing software entities (e.g., Egged and Balzer, 2006; Hu et al., 2008; Huhns et al., 2005; Jureta et al., 2007; Maamar et al., 2005; Motahari Nezhad et al., 2007; Santofimia et al., 2008). However, most of the existing approaches in the literature are centralized and rare of them takes into consideration the quality requirements of the systems and the autonomy of constituent elements.

In principle, the process of assembling Internet-scale software systems by using AGs can be divided into three phases (Medjahed et al., 2003; Rao and Su, 2005).

- Plan or specify the business processes for achieving the system objectives.
- Select appropriate AGs to undertake the activities specified in the processes.
- Make the selected AGs interoperable.

Among these phases, the order of the first two phases is exchangeable, which will result in different assembly strategies. When there are multiple candidates (i.e., AGs) to perform an activity in the system, the business process can be specified in advance so that candidates could be selected based on their properties (especially their QoS) to improve the construction of the software system. On the other side, when every AG can only undertake a special activity totally different from one another, planning the business process ahead will offer opportunities for accomplishing the system objectives in different ways.

However, in the latter case, planning is generally realized by using AI technologies (Rao and Su, 2005). The low efficiency nature of automatic planning technologies makes it infeasible and unpractical to adopt planning technologies to generate the business processes for application systems.

In our opinions, a reasonable and practical solution to the assembly of Internet-scale software systems has to make compromises on the following aspects.

- **Generation of business processes:** For real-world applications, the positions and interaction relationships of participants usually have been specified in advance so that participants could take actions and cooperate with one another in the desired way to achieve the system goals. It is often not a necessity that there is an automatic reasoning mechanism in the systems to generate the cooperation processes for AGs.

- **Selection of AGs:** AGs on the Internet are autonomously evolving and they may quit the environment without notifying others. It cannot be assured that AGs always reply to the requests for services, so selecting AGs once and requesting for services hereafter cannot guarantee the achievement of the systems objectives. Therefore, the selections may have to be carried out all along with the run of the systems and AGs had better be re-selected in time according to their QoS so that the success probabilities and overall qualities of the systems could be improved.

- **Interoperation of AGs:** AGs are expected to be incorporated into different cooperation applications. They are usually focused on the realizations of their computation logic and contrarily they are not hard-coded with application specific cooperation capabilities. When AGs enter application systems to undertake cooperation tasks, they must be informed with application specific cooperation specifications so that they could cooperate with one another in the desired ways.

In this paper, we put forward a decentralized approach to automatic assembly of software systems by using AGs distributed on the Internet. In the approach, the software system is organized as a collection of distributed and interconnected divisions (similar to groups in organizations (Ferber and Gutknecht, 1998) but with more concrete functionalities), in which roles take positions to perform activities of the system and they are concerned about their own businesses as well as communications with others. The overall behaviors of the software system are synthesized from the interactions among the roles. The divisions are responsible for recruiting AGs to undertake the duties of the roles and conducting the interactions of AGs to cooperate in pursuit of the achievements of the system's objectives. Meanwhile, the divisions evaluate the QoS of the AGs according to the returns of the AGs and select appropriate AGs to participate in the activities of the system based on the evaluations.

The main contributions of this work include:

- A new automated approach is proposed for assembling Internet-scale software systems, in which the constituent elements are autonomous.
- A new selection strategy is implemented, by which AGs are selected according to their QoS besides their functionalities.
- A new mechanism supporting the interoperation of AGs is implemented, in which a special type of mobile agents, called delegate, are used to carry the cooperation information specified by roles and facilitate the interactions of AGs.

The paper is organized as follows: Section 2 describes an example system. Section 3 defines the model for Internet-scale software systems and proposes the process of automated assembly of Internet-scale software systems. Section 4 describes the implementation of the approach and experiments for testing the example system. Section 5 compares our approach with some related work. Finally, Section 6 concludes our work and points out some future research directions.

### 2. An example system

To illustrate how our approach is used to assemble Internet-scale software systems, we consider a system that acts as running example throughout this paper.

Considering a university human resources MIS (Management Information System), we are required to assemble it by using AGs on the Internet.

In the department of human resources, there are three divisions:

- Talent management division, with responsibilities for the employment, dismissal/resignation, intra-university transfer, promotion, and retirement of staffs.
- Salary/allowance division, in charge of the determination and adjustment of the salaries and allowances of staffs.
- Personnel files management division, responsible for the maintenance and track of the personnel and payment information of staffs.

In the divisions, every responsibility is assigned to a personnel officer (or a role) specially. For instance, the officers and their responsibilities in the divisions are listed in Table 1.

Besides fulfilling their daily work, the personnel officers often communicate and collaborate with others (including the ones in other divisions) while carrying out their work. For an interaction between persons, it is mostly concerned with when or under what situations officers will communicate with one another. For instance, when a faculty member is promoted, the officer taking charge of promotion will notify the promotion information to the ones responsible for adjusting the promotee's salary and allowance and meanwhile ask the files manager to update the related infor-
Table 1
Roles and their responsibilities.

<table>
<thead>
<tr>
<th>Division</th>
<th>Personnel officer</th>
<th>Responsibilities</th>
</tr>
</thead>
<tbody>
<tr>
<td>Talent management</td>
<td>P11: Employment Information</td>
<td>Input employment info</td>
</tr>
<tr>
<td></td>
<td>P12: Intra-university transfer</td>
<td>Modify department info</td>
</tr>
<tr>
<td></td>
<td>P13: Dismissal/resignation</td>
<td>Modify employment state</td>
</tr>
<tr>
<td></td>
<td>P14: Promotion</td>
<td>Modify position info</td>
</tr>
<tr>
<td></td>
<td>P15: Retirement</td>
<td>Modify employment state</td>
</tr>
<tr>
<td></td>
<td>P21: Salary</td>
<td>Determine salary</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Adjust salary</td>
</tr>
<tr>
<td>Salary/allowance</td>
<td>P22: Allowance</td>
<td>Determine allowance</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Adjust allowance</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Deprive allowance</td>
</tr>
<tr>
<td>Files management</td>
<td>P31: Personnel information</td>
<td>Create personnel file</td>
</tr>
<tr>
<td></td>
<td>P32: Payment information</td>
<td>Modify personnel file</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Hand over personnel file</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Destroy personnel file</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Print personnel reports</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Track salary info</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Track allowance info</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Print payment reports</td>
</tr>
</tbody>
</table>

mation about the promotee. The interactions among the officers are embodied in the communications when the officers perform their responsibilities (Fig. 1).

In the following sections, we will discuss how to assemble the university human resources MIS by using AGs distributed on the Internet, including how to select AGs to undertake the responsibilities of roles in the system and how to make AGs interoperable based on the interaction relationships among roles in the processes of achieving the objectives of the system.

3. Automated assembly process

3.1. Model of Internet-scale software systems

An AG is embodied in the services it provides for outside use and the decisions it adopts to control its service-relevant behavior (Liu et al., 2005). AGs are out of control of users and users can only access and use them as black boxes. It is impractical for AGs to reason about the cooperation processes of achieving the system objectives, so there must be some institutions to specify and manage the cooperation for AGs. Therefore, we view an Internet-scale software system as a coalition of loosely coupled distributed divisions.

- Divisions specify the roles involved in the system and the interactions among the roles. The cooperation processes of the system derive from the interactions among the roles.
- Divisions take charge of recruiting AGs to undertake the responsibilities of the roles and selecting AGs based on the AGs’ capabilities and QoS. Each responsibility is assumed to be undertaken by just one AG at one time.
- The cooperation of AGs is restricted by the interaction relationships of the roles.
- When the participants (i.e., AGs) undertaking the responsibilities are determined, divisions notify the AGs of the related cooperation information (e.g., with whom to interact, when to interact, and what to transmit in the interactions).

The class model for Internet-scale software systems is depicted as follows (Fig. 2).

In the model, a division allocates the responsibilities of roles to appropriate undertaker candidates (i.e., allocating the responsibilities to AGs) based on the degrees of trust to AGs. In some sense, selecting an AG frequently embodies a kind of trust of the division on the AG. We use trust degree (Ramchurn et al., 2004) to specify to what extent the division is willing to select an AG. Generally speaking, an AG is more trustworthy and will be assigned with a higher trust degree if the QoS of the AG is the best. However, QoS is generally represented as probability values (e.g., availability and reliability) or average values (e.g., performance and security) and due to the autonomy, an AG with a high QoS does not mean that the AG will be able to provide high quality services all the time. In addition, the QoS of an AG will be influenced by the application environment in which the AG is situated and the QoS of the AG may not be stable. Therefore, the division will try the services of AGs first and calculate the trust degrees related to AGs based on the trials next. The trust degrees are actually the reflection of the real QoS.
of AGs under their actual application environments. How to maintain the trust degrees related to AGs will be discussed in the process of automatic selection of AGs in the next sub-section. Formally, a division can be described as follows.

\[ \text{Div} = \langle \text{R-set, AG-list, RA-assignment, AG-trust} \rangle \]

- **R-set** is the set of roles involved in the division. A role is specified by its responsibilities and interactions (Zambonelli et al., 2003), i.e., \( R = \langle \text{resp, Interactions} \rangle \).

  resp is one of the responsibilities of the role. The responsibilities of a role denote the functionalities of the role and a responsibility can be simply specified by the signature of a function (including the name of the function, its parameters and their type, and the return value). For example, the responsibility for determining the salary of a new come employee can be described as "determineSalary(String employeeID)".

  Interactions specify how the role communicates with the outside world (i.e., other roles) when it is committed to a responsibility, such as when to start the interactions, what to transmit, and whom to interact with. Interaction = \langle Occasion, Direction, Content, Source/Destination \rangle, in which Occasion is used to specify the situation of carrying out an interaction. We specify two situations for interactions, i.e., "before" or "after" the responsibility is performed; and we divide interactions into two categories according to the communication directions, i.e., "From" a communication source or "To" a destination.

  The content of an interaction is the data transmitted between the source role and the destination and it is specified by a data structure.

  For the simplicity, the communication sources and destinations are specified as the identities of roles.

  For example, after the responsibility of "determineSalary(String employeeID)" is performed, the role responsible for it will transfer the salary information of the employee to the role (referred to as "Payment Information Officer") who is responsible for filing the salary information of employees. Then, one of the interactions involved in the responsibility of "determineSalary(String employeeID)" can be described as "\langle After, To, Salary-Information, Payment Information Officer \rangle".

- **AG-list** is the list of AGs registered in the division, \( \text{AG-list} = \langle AG, location \rangle \). An AG is considered as a black-box and it is defined by its exposed attributes, such as capabilities (specified by service interfaces) and sending/receiving messages in performing its capabilities, i.e., \( \text{AG} = \langle \text{Interface, Messages} \rangle \) and \( \text{Message} = \langle \text{Content, Direction} \rangle \).

  For a role in the role list of a division, the division manages the following information related to the role.

  - **R-set** is the set of roles involved in the division. A role is specified by its responsibilities and interactions (Zambonelli et al., 2003), i.e., \( R = \langle \text{resp, Interactions} \rangle \).

  resp is one of the responsibilities of the role. The responsibilities of a role denote the functionalities of the role and a responsibility can be simply specified by the signature of a function (including the name of the function, its parameters and their type, and the return value). For example, the responsibility for determining the salary of a new come employee can be described as "determineSalary(String employeeID)".

  Interactions specify how the role communicates with the outside world (i.e., other roles) when it is committed to a responsibility, such as when to start the interactions, what to transmit, and whom to interact with. Interaction = \langle Occasion, Direction, Content, Source/Destination \rangle, in which Occasion is used to specify the situation of carrying out an interaction. We specify two situations for interactions, i.e., "before" or "after" the responsibility is performed; and we divide interactions into two categories according to the communication directions, i.e., "From" a communication source or "To" a destination.

  The content of an interaction is the data transmitted between the source role and the destination and it is specified by a data structure.

  For the simplicity, the communication sources and destinations are specified as the identities of roles.

  For example, after the responsibility of "determineSalary(String employeeID)" is performed, the role responsible for it will transfer the salary information of the employee to the role (referred to as "Payment Information Officer") who is responsible for filing the salary information of employees. Then, one of the interactions involved in the responsibility of "determineSalary(String employeeID)" can be described as "\langle After, To, Salary-Information, Payment Information Officer \rangle".

  AG-list is the list of AGs registered in the division, \( \text{AG-list} = \langle AG, location \rangle \). An AG is considered as a black-box and it is defined by its exposed attributes, such as capabilities (specified by service interfaces) and sending/receiving messages in performing its capabilities, i.e., \( AG = \langle \text{Interface, Messages} \rangle \) and \( \text{Message} = \langle \text{Content, Direction} \rangle \).

  For a role in the role list of a division, the division manages the following information related to the role.

  - **The undertaker candidates of each responsibility, i.e., AGs that can undertake the responsibility. An AG is competent with undertaking a responsibility if a capability of the AG can literally match the responsibility and the transmitted content of the AG meets the requirement of the responsibility, i.e., for an AG \( \text{ag} \) and a responsibility \( \text{resp} \) of the role \( r \), \( \text{ag} \in \text{ag-candi} \) if \( \text{competent}(\text{ag}, \text{resp}) \) and \( \text{competent}(\text{ag}, \text{resp}) \) is valid if \( \exists \text{ag-spec} \in \text{ag}(\text{resp} = \text{ag-spec.Interface}) \land \exists \text{spec} \in \text{r-spec.resp} \land \text{matching}(\text{ag-spec.Messages, r-spec.Interactions}) \), where \( \text{matching}(\text{Messages, Interactions}) \) is valid if \( \exists \text{m} \in \text{Messages} \land \exists \text{iact} \in \text{Interactions} \land \text{m.Content} = \text{iact.Content} \land \text{m.Direction} = \text{iact.Direction} \land \forall \text{m} \in \text{Messages} \land \text{m.Content} = \text{iact.Content} \land \text{m.Direction} = \text{iact.Direction} \).

  The current attachments between responsibilities and candidates, i.e., the mapping that defines which AG each responsibility is undertaken by.

  This information is maintained in RA-assignment, i.e., \( \text{RA-assignment} = \langle \text{ag, r, resp} \rangle \land \text{ag-candi} \).

  - For a registered AG in the division, the division manages the following information.

    - The trust degree related to the AG.

    - The responsibility that the AG is in charge of currently.

3.2. Process of automatic assembly of Internet-scale software systems

Since the roles in each division have specified the interactions occurring in the performances of their responsibilities and the interactions will conduct the AGs to pursue the system objectives cooperatively, the process of automatic assembly of Internet-scale software systems is composed of two phases, i.e., automatic selection of AGs and automatic interoperation of AGs for synthesizing the systems.

3.2.1. Automatic selection of AGs

The sub-process for automatically selecting AGs to participate in the systems can be described as follows.

- When an AG enters the system, it first registers itself to the division it knows. The registration information includes

  - its computational capabilities, which are literally specified as the signatures of functions implemented by the AG, and

  - transmitted content, including those incoming and outgoing data transmitted between the AG and others while the AG is performing its functions. The content is specified by data structures as those of roles.

  If the division finds that the AG is incapable of undertaking any responsibility of the local roles in the division, it can recommend the AG to its neighboring divisions.

  After an AG registers in the division, the division will append the AG to the undertaker candidates’ list of a responsibility of a role based on the AG’s capabilities.

  When the division is demanding a responsibility of a role to be carried out, the division will select the most trusted AG to undertake the responsibility.

  Initially, the division trusts all of the undertaker candidates (i.e., AGs) equally and it will allocate the responsibility to an AG by the strategy of first come first allocated.

  Hereafter, the division will tune the trust degrees of the AGs according to the service conditions of the AGs. In principle, the better an AG provides services, the more the division will trust the AG. For different applications and quality requirements, the strategy for the division to tune the trust degrees is generally varied. In Section 4, we adopt a special strategy using a learning
technology to adjust the trust degrees based on the availability and performance of AGs.

- After an AG is allocated with a responsibility, it will be requested for services and be involved in cooperation with others. After the AG finishes providing services, the division will free the AG so that the division could have opportunity to have more trusted AGs to undertake responsibilities to improve the quality of the system dynamically.

The algorithm of selection of AGs is shown in pseudo-code as follows (Fig. 3), in which bold-faced words are variable for denoting different objects.

3.2.2. Automatic interoperation of AGs

After AGs are selected and assigned with responsibilities, they will take up their responsibilities and interact under the restrictions of the divisions.

Although an AG is independent of its environment as well as other AGs, it must interact with others when it is assembled into a software system and cooperates with one another in pursuit of the system objectives. As mentioned before, an AG does not know whom to cooperate with and how to cooperate since it is designed and developed independently. There are two means of enabling an AG to cooperate with others.

- One is to use a centralized coordinator to control the cooperation among AGs. The process engine is a commonly used mechanism (Alonso et al., 2005; Sheth and Verma, 2005).
- The other is to decentralize the cooperation among AGs and let AGs decide locally on the interactions with others. In this case, the cooperation is synthesized from the local behavior of AGs.

We choose the second strategy, i.e., AGs are notified with the cooperation information and subsequently they decide by themselves on whom and how to cooperate with. Thus, an AG can interact with others without the need of relying on a centralized node (e.g., the divisions) so that both the behavior and interactions of an AG will be kept distributed.

However, for AGs, we can neither modify their internal implementation structures nor alter their behavior compellingly. They may have no way to obtain the cooperation information specified by the divisions. On the other side, for the divisions, they must obtain the QoS information about AGs so that they could select appropriate AGs to undertake responsibilities. In other words, the selected AGs should feed back the services status (such as the success of service, the execution time and so on) to the divisions so that the divisions could evaluate the QoS of the AGs according to the feedbacks.

Therefore, we use the following means to realize the interoperation of AGs and the acquisition of QoS information about AGs for the divisions.

- When an AG registers in a division, the division sends a delegate to the site where the AG is located. The delegate takes charge of:
  (i) the invocation of the AG’s services,
  (ii) the transfer of communication messages from the AG to its interlocutors or inversely, and
  (iii) the feedback of the AG’s service status to the division.

The division will dynamically adjust the trust degrees related to the registered AGs based on the returns of the delegates and select appropriate AGs for service invocations subsequently.

- When a delegate is to be sent out, it will carry the interaction information related to the responsibility that the AG undertakes (as shown in Algorithm 3 in Fig. 4).

  Before the division dispatches the delegate, it will first transform the interaction information into the content of FIPA ACL messages and then inform the content to the delegate. For example, for an interaction (represented as aq, aq ∈ info, where info is the interactions specific to the responsibility as mentioned in the algorithm above) for sending information, the corresponding ACL message has the following form, in which the content is another ACL message.

  (inform:sender division_id:receiver delegate_id:content

- When the undertaker of a responsibility is alternated, the division will notify the affected delegates so that the delegates could transfer the communication messages between the AGs and their interlocutors accurately.

![Algorithm 1. Selection of AGs](image)

<table>
<thead>
<tr>
<th>Algorithm 1. Selection of AGs</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input:</strong> AG-list</td>
</tr>
<tr>
<td><strong>Output:</strong> RA-assignment</td>
</tr>
<tr>
<td><strong>Begin</strong></td>
</tr>
<tr>
<td>1. For each capability of an AG in the division, ng ∈ AG-list</td>
</tr>
<tr>
<td>2. For each role in the division, r ∈ R-set</td>
</tr>
<tr>
<td>3. For each responsibility of the role, resp ∈ r Resp-list</td>
</tr>
<tr>
<td>4. If competent(ng, resp) then</td>
</tr>
<tr>
<td>5. add-ng-as-a-candidate(ng, r, resp);</td>
</tr>
<tr>
<td>6. For each responsibility assignment relation in the division, ra ∈ RA-assignment</td>
</tr>
<tr>
<td>7. ra.under-taker – Select-undertaker(ra.resp, ra.under-taker);</td>
</tr>
<tr>
<td><strong>End</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Algorithm 2. Select-undertaker</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input:</strong> resp, ag-candi</td>
</tr>
<tr>
<td><strong>Output:</strong> ag</td>
</tr>
<tr>
<td><strong>Being</strong></td>
</tr>
<tr>
<td>1. Recalculate-Selection-Probabilities(ng-ag-candi); // By Using Formula 6</td>
</tr>
<tr>
<td>2. p = Random(100); // generate a random non-negative integer less than 100</td>
</tr>
<tr>
<td>3. For each ng in ag-candi in which agents are sorted by their selection probabilities degressively, suppose that spng represents the selection probability of ng</td>
</tr>
<tr>
<td>4. if p is not greater than spng then</td>
</tr>
<tr>
<td>5. return ng; // the ng is selected.</td>
</tr>
<tr>
<td>6. else p = p – spng;</td>
</tr>
<tr>
<td><strong>End</strong></td>
</tr>
</tbody>
</table>

![Algorithm 3. Selection of AGs](image)
Suppose that the AG is the new undertaker of the responsibility and "I" represents an interaction of the AG that it makes when it is performing the responsibility, the division determines the interlocutors of the AG as follows:
(a) Find the roles identified by the communication destinations or sources specified in I.
(b) For a found role R' (which may be in another division), look up the interactions related to R' for an interaction I where the source or destination is R and the communication content matches that of I.
(c) If such an I exists, extract the responsibility from the interaction specification and then look into the division's role list for an AG that undertakes that responsibility of R'.
(d) That found AG will be a communication source or destination of the AG.

The pseudo-code listed in Algorithm 4 in Fig. 5 shows how the division finds the affected delegates and notifies the alternation. The notifications of interlocutor alternations are also transformed into the content of FIPA ACL messages. For example, if an interaction of an AG for sending information changes the destination (e.g., AG''), the corresponding ACL message has the following form.

Algorithms were used to formalize the processes of initiating and dispatching delegates.

4. Implementation and experiments

4.1. Implementation structures

As described above, the topology of the university human resources MIS can be depicted in Fig. 6.

- In the system, there involve five types of interconnections.
- Registration links between AGs and divisions. AGs register themselves to divisions so that divisions can assign responsibilities to the registered AGs according to their capabilities and interactions.
- Discovery and recommendation links between divisions. The implementation of discovering AGs borrows the idea of peer discovery in hierarchical P2P systems (Sycara et al., 2004). Besides as the registration center of AGs, divisions are also responsible for publishing (or broadcasting) the registration information for discovery of AGs. Thus, after an AG register in a division, it can be recommended by the division to other divisions when it is not competent with responsibilities of roles involved in the division.
- Delegation links between divisions and delegates. When an AG is selected and assigned with a responsibility by a division, the division will dispatch a delegate to the site of the AG. On the other
side, the delegate will report the service status of the AG back to the division after it tries the service of the AG.

- Invocation links between delegate and AGs. Delegates invoke the services of AGs.
- Communication transfer links between delegates. Although AGs will be selected to undertake responsibilities, they do not interact with one another directly. Delegates will facilitate the communications among AGs via transforming and transferring communication messages between AGs. Because the content transmitted between AGs may be mismatched with that required by roles, delegates will be responsible for eliminating the mismatches and provide unblocked communication channels for AGs.

### 4.1.1. Division

The implementation structure of divisions is shown in Fig. 7. When a division is created, it works as follows:

- **Loads the roles specifications**, including the responsibilities and related interaction information, and **stores the roles information** in the role list (steps 1 and 2). In the current implementation, two responsibilities are identical if they have the same signature and completely involve the same interactions.
- **Records the registration information of AGs**, including the identities and IP addresses of the delegates corresponding to the AGs, **initializes the trust degrees related to the AGs**, and **appends the AGs into the candidate lists** specific to responsibilities based on their capabilities (steps 3, 4 and 5).
- **Selects AGs and allocates responsibilities to AGs** based on the trust degrees related to AGs (steps 6 and 7).
- **Informs the interaction information to the delegates**, dispatches the delegates to the remote sites where the AGs are located, and sends requests for services to the delegates when the responsibilities are planned to perform (step 8).
- **Collects the information** about the services status of AGs fed back by the delegates and **adjusts the trust degrees related to AGs** based on the feedbacks (steps 10 and 11), and before this, the delegates should have invoked the AGs for services and observed the services status of the AGs (step 9).

#### 4.1.2. Delegate

We implement a delegate as a mobile agent like Aglets (Aglet, 2002). A delegate acts like a proxy of a concrete AG. It does not encapsulate the implementation of the AG; instead, it just offers channels for accessing and using the computation of the AG. Then, users can use and interact with the AG by a uniform means through the delegate, which will offer a transparent access to the computation of the AG for users.

The structure of a delegate is shown as follows (Fig. 8). As mentioned above, before a delegate is dispatched to the remote site where an AG is located, a delegate has carried the interaction information related to a special responsibility that the AG can undertake (steps 1 and 2).

If the AG is assigned with a responsibility, the delegate will start to invoke the services of the AG and interact with other delegates.

- When the AG is selected as the undertaker of the responsibility, the delegate calls the services of the AG and feeds back the services status to the division (Steps 3 and 4).
- When a type of “From” interaction is triggered, the delegate waits for data from another delegate and then transfers the received data to the AG. Similarly, when a type of “To” interaction is triggered, the delegate forwards the data from the AG to another delegate (steps 5 and 6).

When AGs undertake responsibilities and interact with one another, there may be interaction mismatches between AGs due to the data structures of the transmitted content (Jiao and Mei, 2006). When a delegate facilitates the interactions between AGs, it must first solve the mismatches and then forward the interaction information to AGs as required. In Jiao and Mei (2006), we implemented an agent-based approach to eliminating interaction mismatches between COTS. By using that approach, a delegate can eliminate the interaction mismatches between AGs as follows:

- When the delegate receives a message from an AG, it extracts the content from the message and forward the content directly to the destination delegate;
- When the destination delegate receives the transmitted content, it first unpacks the content into data items according to the required content of the destination AG and then buffers the data items;
- When the destination delegate buffers all of the data items specified in the required content of the destination AG, it first repacks the data items as the required content of the destination AG and then forwards the content to the latter.

#### 4.2. QoS evaluation and selection of AGs

In our current implementation, divisions select AGs based on the availability and performance (or execution time) of AGs. Corre-
spondingly, the higher an AG’s availability and performance is, the more the divisions trust it.

4.2.1. Adjustment of trust degrees

Divisions adopt the following computation, which is based on the simulated annealing algorithm, to adjust the trust degree related to an AG.

\[
T_{\text{AC}} = e^{-\alpha \cdot \text{Utility}}
\]

(1)

The computation for the trust degree is defined as an exponential function so that the trust degree could be converged fast and consequently divisions could locate and select the AGs with the best availability and performance to undertake responsibilities as quickly as possible.

- \(\alpha\) is a constant for enlarging the influence of the successful invocations and meanwhile for restricting the range of the trust degree.\(^1\)
- Utility is the utility function over a collection of quality properties of the AG.

\[
\text{Utility} = \sum_{i=1}^{n} \omega_i U_i
\]

(2)

\(U_i\) is the utility functions over a special quality property of the AG whilst \(\omega_i\) is the weight of that quality property in the overall utility.

- Specifically, for availability, suppose that SuccessRatio is to record the invocation success ratio among a fixed number of invocations of the AG’s services, the corresponding utility of the availability of the AG is defined as follows, in which the logarithm function is used to ensure that the utility turns large quickly when SuccessRatio becomes high.

\[
U_{\text{avail}} = - \ln(1 - \text{SuccessRatio})
\]

(3)

- For performance, suppose that BestET and WorstET are to represent the scope of the allowed execution time of a responsibility and ExecutionTime is the actually reply time of the AG, the utility of the performance is computed as follows.

\[
U_{\text{perf}} = \begin{cases} 
0 & \text{if ExecutionTime} < \text{BestET or ExecutionTime} > \text{WorstET} \\
- \ln(1 - \frac{\text{WorstET} - \text{ExecutionTime}}{\text{WorstET} - \text{BestET}}) & \text{otherwise}
\end{cases}
\]

(4)

- HotServiceIndicator is an indicator to show how hot the AG is. The hotter the AG is, the more frequently it has been selected by the division. However, when the AG is too hot and it is selected too frequently, it may obstruct the system in trying new opportunities of requesting for services from other AGs. HotServiceIndicator changes as follows.

\[
\text{HotServiceIndicator} = \begin{cases} 
\beta^2 / 2 & \text{if HotServiceIndicator} \geq \beta^2 \\
\text{HotServiceIndicator} + 1 & \text{Otherwise}
\end{cases}
\]

(5)

\(\beta\) is a constant denoting the minimum invocation times\(^2\) that will affect the trust degree related to the AG.

4.2.2. Selection of AGs

Generally speaking, the improvement of the overall quality of the system can be ensured if the divisions always select the most trusted AGs. However, when a new AG registers to the division, the division may assign a low trust degree to the AG since it has not used the AG’s services yet, and this will obstruct the division in trying new opportunities of requesting for services from new AGs with high quality. Therefore, it had better take different tactics in selecting AGs in front of different quality requirements.

In our current implementation, we are concerned about the availability and performance of the system and accordingly we adopt a selection strategy which assigns high selection probabilities to AGs with high QoS. Suppose that there are \(N\) candidates (i.e., AGs) for a special responsibility of a role in the division, the division will use a distribution probability for selecting AGs based on the following calculation.

\[
P_i = \frac{T_i}{\sum_{j=1}^{n} T_j}
\]

(6)

where, \(T_i\) is the current trust degree related to the \(i\)-th AG and \(P_i\) is the corresponding probability of selecting the AG. Thus, the larger the trust degree related to an AG is, the more probably it will select the AG to undertake the responsibility and be requested for services (as shown in Algorithm 2).

Furthermore, when a selected AG fails to undertake a responsibility, the division can retry at most \(N\) times to select another AG if the QoS of the system is tolerable (for instance, the subsequently selected AG can perform the undertaken responsibility in time as specified by the system’s performance requirement).

4.3. Experiments

4.3.1. Settings

We build an experimental system by using the JAVA programming language to simulate the university personnel MIS described in Section 2, in which delegates are implemented as mobile agents by extending the Aglets (Aglet, 2002) development framework.

- In the system, there are 3 divisions deployed at three sites and nine roles involved in the divisions. The roles have totally 19 responsibilities and the total number of interaction relationships among the roles is 20.

\[^1\) \(\alpha\)’s value actually represents the degree of “confidence” in predicting the future QoS of the AG so it had better be variable in different environments. Generally, the value could be enlarged in a stable environment since the AG’s QoS does not change frequently (so that it is more predictable); contrarily, the value should be reduced in an unstable environment.

\[^2\) To avoid being trapped into a local maximum, the division should be encouraged to try different ACs. That is, when the division gains enough invocation feedbacks from an AG and has turned familiar with the AG’s QoS, it had better leave some opportunity for trying those unfamiliar ACs.
In addition, the parameters in the formulas mentioned above are set as follows:

- $\alpha$ in formula (1) is an empirical value designated to 4.6 in the current implementation.\(^3\)
- $\beta$ is also an empirical value and set to be 40.\(^4\)
- When the overall quality of the system is decided by the availabilities and performances of AGs jointly, the weights are set as 0.7 for availability and 0.3 for performance, respectively, in formula (2) considering that the personnel MIS is more concerned about its availability.

Further, for studying and observing the property of the system under varied circumstances (as follows) in which the numbers of the registered AGs are different, we purposely deploy a specified number of copies of AGs on the Internet in each run of the system.

- For each responsibility of the system, there is exactly one candidate to undertake it, or
- For each responsibility of the system, there are at most 5, 10, 20, or 50 candidates to undertake it.
- These candidates are deployed on a cluster of servers. The servers are with the same configuration, i.e., with an Intel\textsuperscript{®} Core\textsuperscript{TM}2 Duo CPU E7300 @ 2.66 GHz, 2.66 GHz and 2.0 GB memory.

4.3.2. Experiments and analysis

First, we test the overall execution success ratio of the system under different situations and assumptions. In each experiment, we execute the system 100K times and sample the execution states per 1K times of runs.

1. Considering only the availabilities of AGs under the assumption that the performances of AGs can always satisfy the performance requirements of the system.
2. Considering only the performances of AGs under the assumption that AGs are always available.
3. Considering both the availabilities and performances of AGs.

When just the performances of AGs are considered, the overall execution success ratio of the system will increase as well when there are more selectable AGs. Nevertheless, as shown in Fig. 10, when the number of candidates increases (for instance, $\geq 10$), the success ratio of the system does not rise remarkably (and even worse, for instance when the number of candidates is about 20). The reason for this is that some selected AGs may have wasted too much of the system’s time even though they fail to provide services and consequently the system has no enough spare time to select other AGs to undertake the unperformed responsibilities. Anyhow, the success ratio of the system reaches its best when the number of candidates is about 50. This is because we initiate some AGs with better performance than the existing AGs and the system can always find those high performance AGs for services.

When both the availabilities and performances of AGs are considered, the overall quality of the system is mainly decided by the performances of AGs (as shown in Fig. 11). Our approach adopts a fault-tolerant likewise strategy of selecting responsibility undertaker candidates so the impact of the availabilities of AGs on the overall quality of the system turns relatively less distinct when the numbers of candidates become bigger.

However, as shown in the figure, when there are more candidates (e.g., the number is about 20 or even more) the overall quality of the system turns even worse. The reason of occurrence of this phenomenon may be because there are many candidates with very close trust degrees and they are allotted with almost equal selection probabilities. As a result, the system may spend much time on altering the undertakers of the responsibilities among these candidates and it makes the quality of the system fluctuant more frequently than normal.

We also evaluate the impacts of methods of calculating the utilities and adjusting the trust degrees on the system’s quality (Fig. 12). We compare the current calculation with the simplest calculation methods, i.e., the utility of an AG’s availability is calculated by the probability of successful replies to requests for the AG’s service (i.e., $U_{\text{avail}} = \text{Success Ratio}$) whilst the utility of the AG’s performance is simply computed as the fastness of the actual reply time of the AG (i.e., $U_{\text{perf}} = \frac{\text{WorstET} - \text{Execution Time}}{\text{WorstET} - \text{BestET}}$). In addition, we also compare our current method of adjusting the trust degrees (as shown in formula (1)) with another method in which the HotServiceIndicator will increase monotonously and the computation of a trust degree is not applied to the logarithm function.

When the calculations of the utilities do not use the logarithm function as those we do in formulas (2) and (3), the system has to take a longer time to find the most available AGs for services. Nevertheless, when the system finds the most available AGs, the overall quality of the system is more stable than that when the logarithm functions are used in calculations of the utilities.

In addition, when the value of HotServiceIndicator increases monotonously, the system can gain a high quality fast than that
when we use the formula (4) to calculate the value of HotServiceIndicator. However, although there are new AGs with better quality joining in the system, the system cannot find them and request for services from them so the overall quality of the system cannot be improved.

Second, we test the extra-communication overhead resulted from the dynamic selections of AGs in our approach. In the system, AGs communicate with each other according to the interaction specifications defined by responsibilities and the communications are compulsory. The delegates will facilitate the communications among AGs. On the other side, the divisions notify the alternations to the delegates when they re-select AGs to undertake responsibilities, and the delegates also feed back the services status of AGs to the divisions, both of which will result in extra-communications. Therefore, the extra-communication overheads of the system are computed based on the numbers of messages transmitted between the divisions and the delegates.

We compare the extra-communication overheads of the system under different situations when the numbers of undertaker candidates of a responsibility are varied. As shown in Fig. 13, at the beginning runs of the system, the system is in the trial phase of finding high quality AGs and the overheads are relatively higher. After a period of trials, the system will find out those undertakers (i.e., AGs) with high quality and it will be inclined to select those AGs for services hereafter. As a result, the alternations of undertakers will take place less frequently and the overheads of the system will decrease consequently.

On the other side, with the numbers of candidates become large, the frequencies of re-selecting undertakers of responsibilities turn higher so the overheads of the system will increase.

Third, we test the scalability of the approach. The cost of assembly of the system mainly consists of two parts, one is that the divisions of the system reason about the competence of AGs with the responsibilities of the roles and assigns the responsibilities to undertaker candidates (as described in Algorithm 1), and the other is that the divisions notify the undertaker alternations of the responsibilities among delegates (as described in Algorithm 4). Considering that the numbers of the responsibilities, roles, and divisions of the system are relatively stable and the divisions are just responsible for evaluating the QoS of AGs and selecting AGs to undertake responsibilities, the increment of the cost of assembly of the system is resulted from the increase of the number of AGs by and large. The cost of assembly of the system is reflected by the extra-communication overheads to some extents.
Therefore, we compare the average increments of the cost of assembly of the system under the situations that the number of responsibility undertaker candidates changes from 1 to a relatively large number (e.g., 100). As shown in Fig. 14, the cost of assembly of the system does not change much when the number of AGs increases, and the number of extra-messages transmitted between the divisions and the delegates is less than two times of the total number of interaction relationships among the roles plus the number of the responsibilities of the system. This implies that the cost of assembly of the system is by and large decided by the complexity of the interactions among the roles whereas the number of AGs has no essential impact on the cost of assembly of the system. This further indicates that our approach has a good scalability.

In the experiments, while performances are considered, the costs of assembly of the system decrease though the selection strategy become more complicated. In the calculations of the costs, once the system fails, we stop selecting other AGs to undertake responsibilities and extra-communications will not happen any more. As a result, the average costs of assembly of the system will turn down with the decrease of the execution success ratio of the system.

5. Related work

Currently, there is much work concerned with the composition of Internet-scale software entities.

For instance, Egyed and Balzer (2006), Jiao and Mei (2006), Motahari Nezhad et al. (2007) study the automated or semi-automated means of integrating heterogeneous components. Alonso et al. (2005), Charif and Sabouret (2005), Maamar et al. (2005), Rao and Su (2005), Sheth and Verma (2005) focus on their studies on the automated composition of Web services and (Hu et al., 2008) surveys the commonly used methods for automated Web services composition.

However, in most of existing work, software entities are regarded to be passive and they are supposed to be capable of providing services continuously.

In Ben-Shaul et al. (1997), a programming framework for composing Internet-based distributed applications called HADAS takes into consideration the autonomy of the computation sites. In the framework, a special component referred to as “ambassador” is designed to support the adaptive invocations of services of components. An ambassador is a representative of a component deployed on a remote site and it has two main purposes: (1) they serve as an adaptive remote reference and (2) as an interoperability handler that can transform the input from the client into a format expected by the component.

An ambassador is similar to a delegate in our approach to some extent. Differently, an ambassador is sent by a component to the location of the client and is responsible for forwarding the client’s invocations to the component whilst a delegate is sent by a division to the location of an AG and is responsible for facilitating the interactions of AGs. Each ambassador must be programmed specifically to a component and an ambassador can only transform the input specific to the component it represents; whilst all delegates are general-purpose agents sharing the same implementation and their capabilities are individualized by the interaction information that they carry, so a delegate can be reused to support the interoperability of different agents automatically.
In MAS, there is also much work on the web services composition based on agents (e.g., Maamar et al., 2005; Santofimia et al., 2008) and Huhs et al. (2005) discuss the research directions in that area. However, current work is still focusing on the composition of functionalities of software systems. As an exception, Sugawara et al. (2006) study how local agent strategies for selecting cooperation partners impact on the total performance of the system. However, that work does not consider the performance constraints of the system and agents in the system are supposed to be always available. In our work, the system is considered to be successful executed only if all selected agents are available and their performances meet the requirements of the responsibilities.

To select software entities for composition, auctions are useful techniques. Pauronby et al. (2007) use CNP (the Contract Net Protocol) for Web service negotiation and selection. Moon et al. (2008) also adopt CNP as the negotiation mechanism to deal with the component selection problem. However, in CNP, it is very hard to specify the auctioned tasks and define the profit evaluation functions for biddings. Existing approaches based on CNP are usually specific to applications and the profit evaluation functions are ad hoc. In our approach, the divisions evaluate the QoS of AGs according to their experiences on trying the services of AGs. Our approach coincides with the characteristics of autonomous software entities on the Internet (i.e., their QoS is unforeseeable and uncertain) and is easier to adopt since it need not define evaluation functions specific to applications.

6. Conclusions and future work

This paper describes an automated approach to constructing Internet-scale software systems based on autonomous agents distributed on the Internet. In the approach, (1) systems are specified by distributed and interconnected divisions and interactions of roles involved in divisions, (2) AGs participate in systems by undertaking responsibilities of roles, (3) divisions dispatch delegates carrying with interaction information to the sites where AGs are located, and (4) delegates are responsible for facilitating the interactions and interoperability of AGs so that AGs could interact with one another in accordance with the interaction requirements of roles to carry out the tasks of systems cooperatively.

While selecting undertakers of responsibilities of roles, the divisions dynamically evaluate the QoS of AGs based on the services status of AGs fed back by the delegates. Currently, the approach takes into account two kinds of QoS of AGs, i.e., availability and performance. As illustrated in the experiments, the approach can improve the overall qualities of systems remarkably and meanwhile satisfy the performance requirements of systems. The experiments also show that the approach is with a good scalability and offers a simple and feasible means of constructing Internet-scale software systems.

In this work, the main concerns are about how to select appropriate AGs and how to support transparent switch among interacting AGs. Therefore, in the experimental system, what AGs will participate in the system and what responsibilities they will undertake are determined in prior by the developers. For the concerns with discovering AGs and making heterogeneous AGs interoperable are not described in detail in this paper. In Sun et al. (2009), we discussed a P2P-wise approach to the registration and discovery of AGs, and in Jiao and Mei (2006), we put forward a method to supporting the interoperation of heterogeneous components, among which there exist even interaction mismatches.

In the experiment, we set the values of the parameters according to our network environments and experiences though we did not display compare the different experimental results when the parameter settings are varied. In the future, we will integrate some learning techniques to make the parameters adaptable to the changes of environments.

In the current implementation, AGs are regarded as completely independent and out of control black boxes and AGs are passively assigned with tasks. At the next stage, we will probe into the autonomous behaviors of AGs and make the approach more intelligent, for instance, allowing AGs to have their preferences of undertaking responsibilities and decide by themselves on whether or not to undertake responsibilities based on the costs and profits.

In addition, the feasibility of the delegate-based approach is under the assumption that the connectivity of the network is robust and delegates are trusted by the host sites where AGs are located. In the future, we will extend the approach to be able to adapt to unstable Internet environments. In our current implementation, the interactions involved in carrying out responsibilities are still very simple, i.e., all interactions are assumed to occur before or after responsibilities are performed. At the next stage, we will extend our approach to enable delegates to support the interoperation among agents using more complex interaction protocols.

Furthermore, we had to admit that the AG-based approach may not be the best choice for implementing the MIS system mentioned above. Nevertheless, the case study is used just for illustrating that the automated assembly approach proposed in this work is feasible and meanwhile it can improve the overall quality of application systems automatically. Currently, we are studying the methodology for the development of Internet-scale software systems by using AGs and applying the approach to more applications.

Acknowledgements

This paper is partially supported by the National Basic Research Program of China (973) (2005CB321805), the National Key Technology R&D Program of China (2008AA012139), and the National Natural Science Foundation of China (60773151).

References


Mei, H., Huang, G., Zhao, H., Jiao, W., 2006. A software architecture centric engineering approach for internetware. Science in China (Series F) 49 (December (6)), 702–730.


