Abstract - We developed one stochastic model for intelligent selection of software components. The selection is done using a XML file containing the most relevant characteristics for each component. XML file has an extra field called "pheromone", which is a concept taken from ant colonies systems. This algorithm can be used for component, service and resource selection; this is possible because our model is general enough for been replicated with different types of requirements.

Key-Words: - Software Components, Selection Algorithm, Artificial Intelligence, Ant Colonies.

1 Introduction
Nowadays the system development is more oriented to reuse software components, which has made increase the number of available component repositories in the Internet [9], [10]. The goal of component use is the division of systems in pieces of code that can be used in more than one software development. The development and adoption of components is increasing with several on-line companies which sell software components and related services. These companies not only allow developers to buy their own components, but also act as intermediaries, providing access to third-party artifacts and related products [29], [30]. Typically, vendors maintain a large inventory of components written in a variety of languages, which are intended for use on different platforms and for different application domains. All of the components are stored in a central repository from which a buyer can buy one or more components. Although there are several component vendors [10], most of these on-line sites have major limitations in the support they provide for searching and finding components. Usually, the components are cataloged with brief descriptions that can be searched through keywords. While this is a good starting point, keyword based searching is not efficient because it often results in too many or too few hits. It may also retrieve components that are completely unrelated [29], [30]. The search of one component is not an easy task, many type of search algorithms have been proposed in literature, some of them are based on hit numbers and popularity [26]. This work tries to use the most relevant characteristics of components like functional requirements described by software developers, in order to make decisions in an accurate way, not only for software component selection but also, in a short time, for any kind of service selection, this is possible because the algorithm is very general and could be used in parallel or distributed computing, when services or processors have to be selected. Our selection algorithm is inspired in the behavior of ant colonies and ant systems (AS) which offer an alternative way of designing “intelligent” systems, in which autonomy; emergence and distributed functioning replace control, preprogramming, and centralization [5], [7]. This paper is organized as following, the second part explains the theoretical aspects of Ant Systems (AS) and components, the third part explains the problem formulation, the fourth part shows our results and finally our conclusions.

2 Theoretical Aspects

2.1 Software Components
A software component is a self contained entity that interacts with its environment through well-defined interfaces (provide services and require functionalities to be provided by other components). Motivations behind component technology are: Independent evolution of application parts, enhanced flexibility, adaptability, and maintainability of software systems. Component-based application development depends on adherence to restricted plug-compatible interfaces and standard interaction protocols [6]. Because of marketplace of software components is highly competitive, in the last years many research have been oriented to software components, how to look for them, how to select
them and how to make them work together as an integrated piece of software [25].

When a component provides services related to aspects of other components, and at the same time can require services from other components, the concept of dependency appears. That is components can have dependency with other components, and this is an important issue to take into account when a component will be selected. Other important concept is information retrieval (IR), which is essentially the process of the content-based, goal-directed extraction of relevant text documents, or more general, assets from large collections. This concept is very important for component selection, because selection, searching and use of components are based on this information. The overall goal of IR is to deliver an exhaustive sub-collection of all single relevant assets (i.e., independently and individually contributing to the user’s goal) [14]. There are four primary functional aspects [21]: (1) interface: Specifies the points where a component interacts with other component, (2) static behavior: Describes the functionality of a component discretely (3) dynamic behavior: Provides a continuous view of how a component arrives at different states and (4) Interaction protocol: Provides an external view of the component and its legal interactions with other components. On the other hand non functional aspects are related to friendly interfaces, usability, flexibility, cost, legal issues among others [27]. The selection of the right component for a system must be done carefully, generally in the planning period of the software development cycle, if the selection is well done; the fail rate is expected to decrease [15]. Many works have been developed for management of software components, an initiative called trusted components was launched; the main idea of this is: (i) to define criteria against which to assess components (so called component quality model) leading to the qualification of existing components and (ii) to enable the component production with fully proved corrected properties by creating appropriate frameworks for specifying these properties and tools for automatic verification [24], [29], [30]. Despite promising results, the approach is still relatively new and further efforts are required [24]. Agora for example, is a prototype being developed by the Software Engineering Institute at Carnegie Mellon University [25]. Other research is presented in [28] its main goal is to provide access to components that can be published at the Web, to be retrieved, and reused in all phases of an application development within a given domain. An approach to intelligent query and component retrieval for web-based repositories is presented in [29], [30].

2.2 Ant Colonies
This Algorithm is inspired by Ant Colonies (AC), which is one of the most used models for describing collective behavior is the Ant System model that is based on observation of ants [11]. Ant System Models (ASM) can be applied in many areas like telecommunication, robotic, patterns recognition, transportation and military applications [19]. The main idea of ASM suggests that N agents (ants) in a colony are cooperating to achieve some goal. The agents use simple rules to govern their actions, and via interactions of the entire group, the collective achieves its objectives. A type of self-organization emerges from the collection of actions of the group [19]. Ants solve problems in a flexible, adaptable, robust, and decentralized way. This definition includes any attempt to design algorithms or distributed problem solving devices inspired by the collective behavior of ant colonies [7]. Many ant species have trail-laying trail-following behavior when foraging: Individual ants deposit a chemical substance called pheromone when they move from a food source to their nest, and foragers follow such pheromone trails in order to find the most efficient path to a food source [7]. Each ant can be considered an agent (software agent), these agents are involved into groups, called colonies, where they do a cooperative work. The agents are able to process information, modulate their behavior according to the stimulus and take the best decision based in the environment information. AC has been used in many different ways and has inspired techniques, like particle swarm optimization, ant colony optimization, culture algorithm, and others [20]. Sign-based communication is highly developed in ants. Ants use highly volatile chemicals called pheromones to provide a sophisticated signaling system. Ants foraging for food lay down quantities of pheromone marking the path that it follows with a trail of the substance. An isolated ant moves essentially at random but an ant encountering a previously laid trail will detect it and decide to follow it with a high probability and thereby reinforce it with a further quantity of pheromone [20]. One of the main works in Ant Systems has been proposed by Dorigo, Maniezzo and Coloni [12], [13] they have proposed the artificial ant system and introduced three types of ant algorithms [12], [13], [18] ant-density, ant quantity, and ant cycle, which can be used to solve different combinatorial optimization problems. These algorithms have been used to solve the quadratic
assignment problem, the traveling salesman problem, the vehicle routing problem, the connection-oriented network routing, dynamic combinatorial optimization problems, etc [2], [3], [4], [7], [8], [18].

3 Problem Formulation

This work proposes a stochastic selection algorithm for software components, which is based on Ant System Colonies; this allows selecting the best component from a group that has been found in Internet according to a set of initial specifications. The selection algorithm uses pheromone tracks in order to identify the best components adding pheromone each time that the component performance is tested. The repositories of software components are seen as food source and each agent as an ant, then, each agent finds repositories and components and select them according not only to pheromone but also initial functional and non functional requirements. Agents update pheromone values for each component. A characterization file (component profile), is used to store information associated to each component. This file will have a group of both dynamic and static data, the static data is not going to change, it is referred to the component name, identification, etc, and the dynamic data can change as many times as necessary. Data contains fields like technical platform, location, programmers, performance associated to a specific technical platform where the component is running. Each component may have more than one sub-profile for each dynamic environment; a sub-profile is related to the domain where the component is loaded, each profile is associated to an amount of pheromone that reflects the component performance in a determined domain, and each component can have an undefined number of sub-profiles. The different sub-profiles are stored in the same XML file (Fig. 1). After a component is selected, rejected or used, the component pheromone is upgraded; pheromone field can be incremented or decremented (evaporated) according to the component performance. Characteristics for a component evaluation have been chosen using factors related to production time, cost rating, product quality and development risk [15], [16]. The initial wished requirements, like component characteristics related to functional and non functional aspects must be well defined in order to generate an XML file that identifies the ideal profile for the component that is going to be selected. In this way, when the group of analysts and programmers search a component, an ideal component profile is generated; this ideal profile will be compared with the different sub-profiles of the found components in order to select the best choice (Fig. 1). Our algorithm assumes that exists as many agents as the user decides, the number of agents is not tie to the number of components to select, and each agent will select as many components as the programmer requires.

![Figure 1: Component Profile – file](image)

The final software development will be done by programmer team after each agent completes all the process and it has all the required components. Programmer will chose the best solution given by the different agents, as more agents more different possible solution. The general constraints for our algorithm are:

- A component to select should always exist.
- Each component has at least one profile into an XML file; the file should allow the incorporation of sub-profiles.
- It is assumed that the initial requirements are exact information about components.

The calculation of matching between the ideal component and the selected one is given by:

$$W_{ij} = 1 + \sum H_j \cdot \sum N_{ij}$$

From a list of $i$ components to search, there is a $W_{ij}$ that is the matching degree for component $j$ which has been located in repository $i$, $j$ has sub-profile $u$, and it has been preselected by agent $k$ that has to select the component more similar to the ideal component $C_j$. $H$ identifies the ideal characteristics ($C_j$) that represents the wished profile; $N$ represents real characteristics of the pre-selected component (component sub-profile). If $W_{ij}$ is close to 1, means that the ideal characteristics are very similar to the real characteristics, while less value of $W_{ij}$ better will be the matching between the ideal component and the selected one. This algorithm is based on:

1. Each agent has a tabu list, a memory that will
store location of selected components; this memory is empty at the beginning of the selection work.

2. The pheromone will be updated when agents finish their work and components are selected and used.

The probability for an agent \( k \) that search a component \( i \), selects a component \( j \) with sub-profile \( u \), into the repository \( l \), among a set of \( s \) components with sub-profiles \( n \), in repositories \( r \), into an iteration \( t \), is coming by:

\[
P_{lju}^{kt}(t) = \frac{[Y_{lju}(t)] [W_{lju}^{ki}]^{-1}}{\sum_{rsn} [Y_{rsn}(t)] [W_{rsn}^{ki}]^{-1}}
\]

(2)

For all \( C_{sn} \) that belongs to conjunct \( CC^{iu} \) which is the set of components whose \( W_{rsn}^{ku} \) are close to 1. The pheromone track is represented by \( Y_{lju}(t) \). It is important to note that even when (2) is used for different algorithm iterations, the value of \( P_{lju}^{kt}(t) \) can be different for each agent evaluating the same component. This will depend on the evaluation of a particular agent, of its set \( CC^{iu} \). After one component selection is completed and used, each agent \( k \) deposits pheromone \( \Delta Y_{lju}^{kt}(t) \), this value depends on the system performance \( R^{ki} \) proposed by agent\( k \), which content all the selected components that have been selected by agent \( k \):

\[
\Delta Y_{lju}^{kt}(t) = \sum_{Si_{ku}=C^{iu}_{kt}} (W_{lju}^{ki}(t) * R(t)^k)^{-1}
\]

(3)

Where

\[
R = f (TE, M, ND)
\]

(4)

\( R \) is a performance function coming by the system execution time (ET), number of component dependencies (ND) and used memory (M). We assume the selection cycle is complete when an agent selects all required components group and programmer team test the system proposed by the agent. This model needs to calculate both, positive and negative feedback, the last one is called pheromone evaporation, this will be introduced through an evaporation coefficient \( \alpha \). Each component pheromone is updated using:

\[
Y_{lju}(t) \leftarrow  (1 - \alpha) * Y_{lju}(t) + \Delta Y_{lju}^{kt}(t)
\]

(5)

The initial component pheromone is assumed as a constant positive \( Y_{0} \) that represents a homogeneous pheromone distribution on a time 0. \( \alpha \) is a constant that controls the pheromone trail evaporation.

In order to choose the component selection process, we have defined the next transition rule:

\[
S_{ki}^{*} = \arg \max_{r \in CC^{iu}} \left\{ P_{r}^{ki} \right\} (J (RandomValue))
\]

(6)

Here, \( P \) is coming from Eq. 2, \( S_{ki}^{*} \) is component \( i \) selected by agent \( k \), from a group of components \( CC^{iu} \). \( J \) is a random value. Equations (2, 5, 6) are based on the works of Dorigo [7], [12], [13]. Our macroalgorithm is:

1. Definition of software architecture: definition and identification of \( k \) required components.
2. Create \( n \) selection agents.
3. Each agent makes selection of all the required components using the next procedure:

Procedure:

1. for \( i = 1 \) to \( K \) (\( K \) is the number of components required)
   1.1. Search components that are close to required component \( i \) using any search engine (1)
   1.2. Select one of them using (6)
2. Evaluate performance (\( R^{k} \)) of the proposal systems, which use components selected by each agent \( k \).
3. Update pheromone track for each component sub-profile.
4. Select the best proposed solution found by one of these agents.

4 Problem Solution

Two repositories were considered: freshmeat.net [22], and SourceForge [23]. We have done a copy of these repositories to test our algorithm. We assumed that three specific components had to be selected, some functional and non functional characteristics are wished, and four agents that are going to make the selection. See Table 1, Table 2, Table 3 and Table 4 for results. The initial amount of pheromone was incorporated randomly; we run our algorithm using 5 functional and non functional requirements. Each agent selected a group of components according to the initial requirements. Even when more than one hundred tests were done, we are showing the most significant results [1].

<table>
<thead>
<tr>
<th>Component</th>
<th>First 3 Components chosen</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>R1,7; R1,1; R2,6</td>
</tr>
<tr>
<td>B</td>
<td>R2,19; R1,9; R1,12</td>
</tr>
<tr>
<td>C</td>
<td>R1,14; R1,4; R2,7</td>
</tr>
</tbody>
</table>

Table 2: Selection Results - Agent 2

<table>
<thead>
<tr>
<th>Component</th>
<th>First 3 Components chosen</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>R2,5; R2,6; R1,7</td>
</tr>
</tbody>
</table>
Table 3: Selection Results - Agent 3

<table>
<thead>
<tr>
<th>Component</th>
<th>First 3 Components chosen</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>R1,7; R1,1; R2,6</td>
</tr>
<tr>
<td>B</td>
<td>R1,13; R1,19; R2,15</td>
</tr>
<tr>
<td>C</td>
<td>R1,17; R2,7; R1,14</td>
</tr>
</tbody>
</table>

Table 4: Selection Results - Agent 4

<table>
<thead>
<tr>
<th>Component</th>
<th>First 3 Components chosen</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>R1,7; R1,1; R2,3</td>
</tr>
<tr>
<td>B</td>
<td>R2,15; R1,11; R1,12</td>
</tr>
<tr>
<td>C</td>
<td>R1,14; R2,7; R2,1</td>
</tr>
</tbody>
</table>

Where Ri,j means component j in repository i.

Generally, popular search algorithms consider the following non functional characteristic in order to sort the results [22], [23]: Popularity. The popularity score superseded the old counters for record hits, URL hits and subscriptions. That is, they provide an initial order of the components using popularity score. In our Algorithm even when the most popular components generally are selected, they are not the first choice in all cases, they were located in the first twenty places. We have seen that most popular component is not always the best and it is not going to have better performance or better matching with initial user requirements. On the other hand vitality depends on the announcements that are not always related to performance or technical integration one component can have a high vitality even when it doesn’t have a good performance; finally component rating depends on votes given by users, who have worked with components in a specific domain, and very particular conditions. Our algorithm uses not only initial user requirements (functional and non functional) but also takes on account the pheromone which is a value that had been incorporated in the component XML file and it is associated to the component performance according to a specific domain. Even when the results are similar among the different ways of sorting (popularity, vitality, rating, our algorithm), in the particular case of our algorithm, for component A for example, component 1 and 7 are first in the list of proposed selection for component A, and they take place 1st and 7th when popularity algorithm is used. Component 1 takes one of the first three places using both algorithms but component 7 is selected as a first option when our algorithm is used. This result is expected because this is the component with more characteristics related to the initial user requirements and more amount of pheromone.

Based on rating, component 8 would be the best choice for selection of a component B, but this component doesn’t have good matching with user requirements and our algorithm does not consider it. Finally the presented algorithm gives better result because generates a list of components that not only accomplishes the initial user requirements but also uses pheromone upgrades giving an approximation related to component performance according a specific domain, independently of component popularity, rating or vitality.

5 Conclusion

This work presents a selection algorithm based on ant colonies that uses a pheromone value related to component performance. Selection and search algorithms based on popularity, vitality or rating are very subjective and not always give to the users the best choices. Our algorithm uses functional and non functional requirement to make the selection; any search engine in order to find a group of components can be used and our algorithm is applied in order to select the best choices. This selection algorithm is based on pheromone values that are upgraded depending on component performance; this gives the opportunity to each found component to be evaluated and its pheromone upgraded. The proposed algorithm can be used with a bigger number of agents which is recommended when a bigger number of repositories are used.

Because of the generality of our model, we propose to test this algorithm with other services related to parallel, distributed and grid computing. It would be very interesting to use our algorithm with naming and trading services, among others criteria. In future works we will show the impact of dynamic composition of components as part of a system architecture that we are working on.

Referentes


[29] Sugumaran V and Storey V. “A Semantic Based Approach to Component Retrieval”. The Data Base for Advances in Information Systems – Summer 2003 (Vol. 34, No.3)