Visualization of the Carrot 2 System

Thomas Laufert

Center for Architectures for Data-Driven Information Processing
University of Maryland Baltimore County
Baltimore, MD 21250
tlaufe1@cs.umbc.edu

Abstract
The Carrot2 (C2) system is a Multi-Agent distributed information retrieval system. In this paper we will describe the visualization tool and discuss the issues involved in visualizing a system, such as C2. We will examine the event ontology which the visualization tool will utilize. We will describe the status of the visualization tool, as well as future directions for the project.

Introduction
The Carrot2 (C2) system is a multi-agent distributed information retrieval system. In C2 the agents act autonomously and each C2 agent handles a particular portion of the collection of information. When the system is started a particular number of C2 agents are instantiated on a number of nodes. The agents then acquire their portion of the information from the Collection Manager. Each agent then computes a meta-data vector for their collection, and then forwards its meta-data to some other C2 agents. Once the system has been setup, queries are given to the C2 agents and they are either processed or forwarded to another (more appropriate) agent.

Since the agents act autonomously, their behavior, the distribution of agents, distribution of collections, distribution of meta-data, and query processing will constantly change. It would be interesting, and extremely helpful (almost necessary) to be able to visualize how the agents act in this environment, (their communication patterns, the forwarding and processing of queries, distribution of meta-data, etc). This paper will describe a system which visualizes the C2 system, where we describe the issues in dealing with a large scale multi-agent-system such as this.

There are many issues which must be dealt with when visualizing such a large scale system. The user must be able to grasp as much (of the important) information as possible. This can pose a great obstacle, since we must work in a limited real estate (a computer screen). The C2 system can have an extraordinary amount of information, and network traffic. To show all of this information on a screen at once would be impossible, and useless to any user attempting to discern that information. We must tackle this problem of limited real-estate.

The first section, the background for the project, will discuss the motivation for our project. Next we will examine related work in the area of visualizing large scale multi-agent systems. Third we discuss our visualization tool. We will look at issues we encountered when implementing the tool as well as various research concerns. Next we will discuss our work briefly, then address the possibility of future work. Finally we will conclude our paper with the outcome of our tool.

Background
The motivation for this project is to provide a way of visualizing how the C2 system functions. This is useful for two reasons: To demonstrate to a large audience the performance of the C2 system at conferences and symposiums. To show to the members of the C2 research group the how the system is performing to aid in debugging the C2 system.

The visualizer must be able to discern which information that is in the system is important. It must then be able to take this information, and in an efficient manner present it to the user. The visualizer must not be a burden on the overall efficiency of the multi-agent-system. The visualizer should also be a modular application, meaning that when new information is desired to be visualized, the visualizer should be able to incorporate the new information relatively easily.

Related Work
As far as we have seen, there has been little written about the problem of visualizing large scale distributed multi-agent information retrieval systems. However, Ndumu, et. al. (Ndumu 1999) discuss the problem of visualizing distributed multi-agent systems. In their work they discuss the two key issues, observing how the agents behave, and assistance in debugging the system. Their work focuses on a visualization tool which could be applied across many different domains where multi-agent-systems could be applied. The tool which
they had developed was split into many different sub tools where a different view of the system could be compared to the other tools. The tasks that were performed by the tools were: to provide a society view which offered a hierarchical view of the relationships of agents, to provide a view of what task a particular agent or set of agents were working together to accomplish, to provide an intimate view of the inner workings of a single agent which the user specifies, to provide control over the agents, and to provide some analysis of the multi-agent-system. This visualization tool has been modified and placed in the ZEUS environment for multi-agent systems.

Szekely, et al. (Szekely 2001) implement a visualization tool which utilizes a punch card system, which is claimed to be able to visualize tens of thousands of agents in a multi-agent System. The punch card system is said to show the behavior throughout its entire lifetime efficiently. This approach to the visualization of a multi-agent-system differs from Ndumu’s by the scope. Ndumu’s tool is not designed to be able to visualize tens of thousands of agents simultaneously. However Szekely’s tool limits the amount of information displayed to the user to various colors, and flashing “lights” on the punch card.

The JADE project is an attempt to create a system which will allow the creation of multi-agent systems to be performed quickly and easily. JADE is a development environment written in java. JADE’s environment incorporates the visualization of the agents into the JADE environment. This allows for an easy and convenient way to design a multi-agent-system, and immediately see how the system is performing. The JADE environment incorporates the following tools, an agent management tool which allows for the stopping and starting of agents either locally or remotely. A DF Manager which allows for the control, visualization, and federation of the DFs which are within the multi-agent-system. There is a Dummy Agent tool which allows for the testing of an agent before an agent is placed in the multi-agent-system. One can see how the dummy agent is performing in the local environment before placing the agent into the multi-agent-system, possibly causing problems with the other agents. There is a sniffer tool which allows for the visualization of message traffic of single or multiple agents. The introspector agent allows the monitoring of a single agent, to visualize its message patterns and its life-cycle. JADE has become increasingly popular recently. This could be because of the visualization aspect for quick development, and debugging.

The RETSINA MOCHA project is an effort to devise an environment for mobile communications network which will run on any platform, anywhere. The heart of MOCHA is the Personal Agent which would perform tasks for humans which they could not already do. The MOCHA project incorporates a visualizer of the system. The visualizer allows the user to see logical communication links between agents in the system, in an iconic way.

With the increasing popularity of the agent research field we have seen the increased popularity of “all-in-one” systems as well. These systems such as JADE, ZEUS, and MOCHA represent only a few of these efforts that are ongoing. Notice how each of these projects incorporate some kind of visualization tool. As more and more people enter the agents field, they will soon realize how large a scope a multi-agent-system can encompass. With this said people will not only desire, but will need, a sophisticated multi-agent-system visualization tool to realize how the system is performing, and to aid in the debugging of their multi-agent-system.

**Carrot 2 Visualization**

The visualization tool which we propose is similar to Ndumu’s tool except we feel that it should be more scalable to visualize numbers of agents in the tens of thousands. The tool will have two main views, one which visualizes the distribution, routing, response, and processing of queries, and one which visualizes meta-data handling. To address the issue of what information should be visualized we have described an ontology which describes which events should be handled by the visualization tool.

The Visualization tool views the C2 system through the information that the Logger agent gives to it. The Logger agent must also adhere to the ontology presented below.

**Ontology**

Figure 1 is a visual representation of the event ontology. Notice how events are either an Administrative event or an Operational event. Those events which have been formalized are included in the diagram. These are the events which we will be describing in detail. They include: registration of agents, routing of queries, receiving a query, forwarding a query, processing a query, responding to a query, and receiving the results of a query. Other events which are not yet formalized include all events dealing with meta-data and the distribution of collections.

![Figure 1: Event Ontology Hierarchy](image)

We will now describe each event, and will describe how the Logger will present this information to the
visualization tool.

**Agent Registration.** Agent registration is an administrative event that takes place for every agent in the C2 system. This event will occur before any other event that deals with the agent in question. The format of the information which the Logger agent will provide to the visualizer is as follows:

```
 event:register:<name_of_agent>:<time_stamp>
```

*<name_of_agent>* is the fully qualified name corresponding to the agent who is now registered, and *<time_stamp>* is the time that the registration was reported.

**Query Routing.** Query Routing occurs in each C2 agent when deciding what should be done with a query it has recently received. The outcome of the routing will determine if the agent will process the query or forward it. The format of a routing event as outputted by the Logger agent is as follows:

```
 event:query_route:<name>:<time>:<query>:<route>
```

*<route>* is a string which contains any decision which was reached about what to do with the query, the options include, but not limited to, process or forward.

**Query Receipt.** This event corresponds to when an agent receives a query. This will be the first event an agent will submit when dealing with a specific query. The format of the event is as follows:

```
 event:query_receipt:<recipient>:<sender>:<time>:<query>
```

where *<recipient>* refers to the fully qualified name of the receiver of the query, and *<sender>* is the name of the sender of the query. *<query>* is the text string of the query which the C2 agents will use to query their corpora for relevant documents.

**Query Forwarding.** Query forwarding occurs in the Carrot 2 system when a C2 agent has received a query, has routed the query, and determined through its meta-data vector that another agent is more qualified to handle the query. The agent then forwards this query to that agent. The format of this event is:

```
 event:query_forward:<sender>:<recipient>:<time>:<query>
```

*<sender>* refers to the agent who is forwarding the query (or the reporter of the event). *<recipient>* is the agent who is the intended receiver of the forwarded query.

**Query Processing.** A query is processed when an agent has received a query, and routed it, and then has determined that it is qualified to handle this query. The agent will attempt to pose the query to its corpus and retrieve a set of relevant documents. The query processing event is formatted as follows:

```
 event:query_process:<name_of_agent>:<time_stamp>:<query>
```

**Query Response.** A query response event occurs after a C2 agent has received a query, routed the query, and processed the query. When an agent responds, it sends back the results to the sender of the query. The format of the event is:

```
 event:query_response:<sender>:<recipient>:<time>:<query>:<result>
```

*<sender>* is the name of the sender of the response (the reporter of the event). *<recipient>* is the name of the receiver of the response (or the agent who sent the query to the responding agent). *<result>* is the actual results found by the responding agent.

**Receiving the Results of a Query.** When an agent receives the results of a query it will throw this event. If the agent reporting this even did not receive this query from another agent, this will be the final event of the query’s life-cycle.

```
 event:query_rec_result:<receiver>:<sender>:<time>:<query>:<result>
```

*<receiver>* is the name of the agent who received the results of the query. *<sender>* is the name of the agent who had responded to the query.

**Other Events.** As of now, the events described above are those events which are formally defined. There are many other messages or events that are present in the system. However, in order to visualize all of the events present in the system would require much more screen real-estate than what is available. This is why the visualization tool, described through the events listed above will visualize only events dealing with queries. Making our tool useful for discovering how queries are distributed, processed, and forwarded, throughout the system.

This could provide useful information when debugging the system. For example, a Carrot 2 agent has just received a query for the Query Manager. Its next course of action is to route this query. You see both events on the screen, receipt and routing, in order. You know by examining the results of the routing decision that the intended action to deal with the query is to forward it to the Carrot 2 agent x. You see the query forward event on the screen, as you expect. Now you expect to see the query receive event for agent x. However, this event is never seen. With this information you now know exactly the point of failure and this information can help aid in the debugging of the problem. This example, albeit simple, is useful for thinking about the possibilities for the aid of debugging in a system such as this. This type of information will let the users see errors such as the one described in a fraction of the amount of time required by trying to read the actual KQML messages which are passed through the system.

**Implementation**

The visualization tool is implemented as a Java application using the Swing tool-set. Our tool reads
events from a file provided by the Logger agent. These events, which were described in the previous section provide all of the information about the system to the visualizer. The visualizer gets no other information about the system from any other source.

The Logger agent receives messages from all agents in the Carrot 2 system which would have events to report that are specified in the ontology presently. Therefore, all agents should send a message to the Logger agent when they have registered with the Agent Name Server. All other events in our ontology involve queries, so the only agents reporting these events will be the Carrot 2 agents, and the Query Agent. Since only a fraction of the messages in the system will result in the transfer of a message to the Logger agent, the overall network traffic will not be significantly affected by the Logger agent. When the Logger agent receives the events from the agents it writes the events to file in the formats specified above.

The visualization tool gets all of its information buffered through the file written by the Logger agent. This allows the visualization of the Carrot 2 system in an offline state, since it can read the file when the system is not running. Once the visualizer has read in an event from the file provided by the Logger agent, it must determine what type of event has been read and how to visualize it. The layout of the visualizer agent consists of the main view panel, which consists of up to four lists representing the nodes. These lists are, in effect, history lists of events that occur on the nodes. The user can of course change which nodes are being visualized on the screen at one time if there are more than four nodes running at one time. There is a control panel which enables the user to rewind, go forward, stop, or start, the playback of the history of the events. There is also an icon panel which gives visual representations for the system agents (Agent Name Service, Collection Manager, and Query Manager). The way that each event is visualized is described below:

**Agent Registration.** Any agent may report a registration event. When the visualizer receives the registration event it will do one of a few things, depending on the agent who has just registered itself. If the agent who registered itself is a system agent, the visualizer will place the appropriate icon in the icon panel. If the agent registered is a node agent, then the visualizer will display a history list corresponding to that node. This way, only those nodes which are running will have history lists associated with it. If the agent which has been registered is a C2 agent the agent’s name will be placed in the history list (corresponding to its node) in the color green.

**Query Routing.** When the visualizer receives a query routing event, the name of the agent will be placed in the history list in an orange color.

**Query Receipt.** When an agent reports that it has received a query, the visualizer will place the agent’s name in its history list in a blue color.

**Query Forwarding.** Query forwarding is visualized by the reporting agent’s name being put in its history list in the color yellow.

**Query Processing.** When an agent has processed a query the visualization tool will place its name in its history list in the color pink.

**Query Response.** Responding to a query causes the visualizer to place the reporting agent’s name in its history list in a red color.

**Receiving the Results to a Query.** When an agent receives the results to a query its name will be placed in its history list in the color black.

**Discussion**

Several issues arose during the development of this visualization tool. The first, and possibly most prevalent, being what information should be visualized. Several drafts were constructed as to what information should be visualized until we formalized the event ontology which is described above.

The issue of what information to visualize is also must be thought about when realizing that there is a limited amount of space provided on a computer screen. Realizing this is where we decided to focus on specified aspects of the Carrot 2 system, rather than trying to visualize the entire inner workings of the system on one computer screen at a time. How would we be able to visualize many (tens of thousands) agents reporting many different events simultaneously? We decided to use different colors to represent the different events in our ontology.

From the beginning of this project we knew that we wanted a visualization tool which could work both while the system was running as well as offline. We achieved this through a logging mechanism offered by a Logger agent.

**Future Work**

We plan to extend this visualization tool to incorporate an augmented event ontology. This new event ontology will incorporate Collection distribution, and Meta-Data distribution. These two new groups of events will require new tools, or a different way to visualize the different style of events.

Also, we plan to incorporate the Logger agent as a component to the visualization tool, to eliminate disk IO during online visualization.

Another possibility is to make this visualization tool work on a different multi-agent-system. This would probably be a more difficult work compared to the previous two possibilities for future work.

**Conclusion**

In this paper we discussed a tool which visualizes the Carrot 2 system. It is designed to allow a user to observe the behavior of agents in the system, as well as to facilitate debugging the system. We propose an event ontology which tells all of the agents in the system which events they should announce to the Logger agent to be visualized. Visualization tools augment the usability of a system, they will become increasingly important to the vitality of a multi-agent-system as designers begin to create extremely large scale
multi-agent-systems. When these large scale systems are deployed the creators must have a way to fix problems easily. With multi-agent-system visualization this is possible. A prime example, in the Carrot 2 system. The Carrot 2 system is an extremely large scale information retrieval system; without the aid of a visualization system debugging can be extremely difficult. However, with the aid of this visualization tool debugging can be significantly easier.

References


Szekely, P.; Rogers, C. M.; Frank, M. 2001. Interfaces for Understanding Multi-Agent Behavior, Information Sciences Institute, University of Southern California.

The JADE Project Web Page: http://sharon.cse.it/projects/jade


