Intelligent Agents Group Project: Group A

Word Count: 1,097 / 1,100

Development Team Project: Project Report

Group A will develop a multi-agent digital forensics system to identify file types on a system and store and report its results. It will determine file types, verify MIME types, summarise text and report the type count. Group members are:

Chris (UK) - Team Leader, Requirements, Functionality, Design
Ahmed (Bahrain) - Literature Review, Challenges, Design Pros and Cons, Developer
Yassir (UK) - Report Writer, Diagrams
Lisa (Vietnam/Austria) - Quality, Compliance Review

Digital forensics has three main phases: acquisition, consisting of interaction with the source medium, analysis, involving inspection and identification of files found there, and presentation of statistics or a description in legalese (Carvey & Altheide, 2011: 3-4). These can be split into more phases - Fig 1:

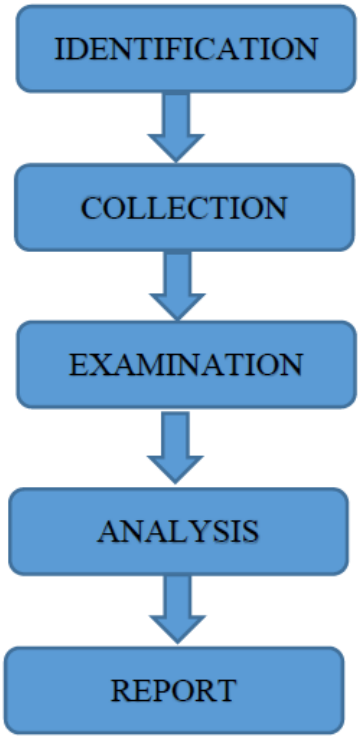

Fig 1: Process Flow (Ghazinour et al, 2017)

Several tools already perform these tasks, including EnCase, a fast utility in use by most security organisations, the open-source Digital Forensic Framework,

Pro-Discover, which includes a “smart agent,” Forensic Toolkit, the command-line Sleuth Kit, etc. (Ghazinour et al, 2017).

The requirement is to collect information about file types, verify that files actually contain data appropriate to their extensions, summarise briefly any text files, archive the results, perform statistical analyses of the number and ages of each file type and report on the results in plain English.

Design and Methodology

Symbolic, reactive and hybrid Python agents with appropriate libraries will perform different tasks, passing on commissives in an agent control language such as KQML or KIF (Labrou & Finin, 1994), running mainly in sequence, each being locked until valid data is passed by the previous one using KQML.

The Requirements symbolic agent receives and interprets the user requirement in plain English, using the spaCy NLP module, passing it to the Search agent in JSON format (drive letter and file mask) within the KQML message - Fig 2:

```
(request
 :sender agentRequirements
 :receiver agentSearch
 :ontology file-search
 :language JSON
 :content
 "{
  \"action\": \"disk_search\",
  \"parameters\": {
   \"drive_letter\": \"C\",
   \"file_mask\": \"*.txt\"
  }
 }"
 :reply-to agentTypes
 :reply-with search_result
)
```

Fig 2: KQML message containing JSON parameters.

The Search reactive agent searches using drive_letter\file_mask, passing the result in JSON (list of file paths/names) to the MIME reactive agent, which loops through all

the files, opening each one and noting the MIME type of its contents. This passes to the Types agent an array of objects, each consisting of a file path\name and its MIME type.

The Types symbolic agent loops through this, parsing out each file's extension and using Python's mimetypes module to check MIME types. It passes to the Text agent the JSON objects with a third key-value pair called RightType, containing True if the MIME and extension match, otherwise False.

The Text hybrid agent loops through the array, opening all files containing text, using the Transformers module to write a one-sentence summary (without commas) in a fourth key-value pair called Summary, which is passed to the Stats, Summary and Archive agents.

The Stats reactive agent builds up an array of file type descriptions, incrementing each type's counter as it loops through the files. It outputs just this array, which will also include a count of extension/MIME mismatches for each type. Simultaneously, the Summary hybrid agent writes a short summary of the combined summaries produced by the Text agent.

The outputs of the Statistics and Summary agents are sent to the Report agent, which uses the Transformers NLG module to produce a report in plain English, while the Archive reactive agent saves the results to a database.

The sequence is illustrated in Fig 3.

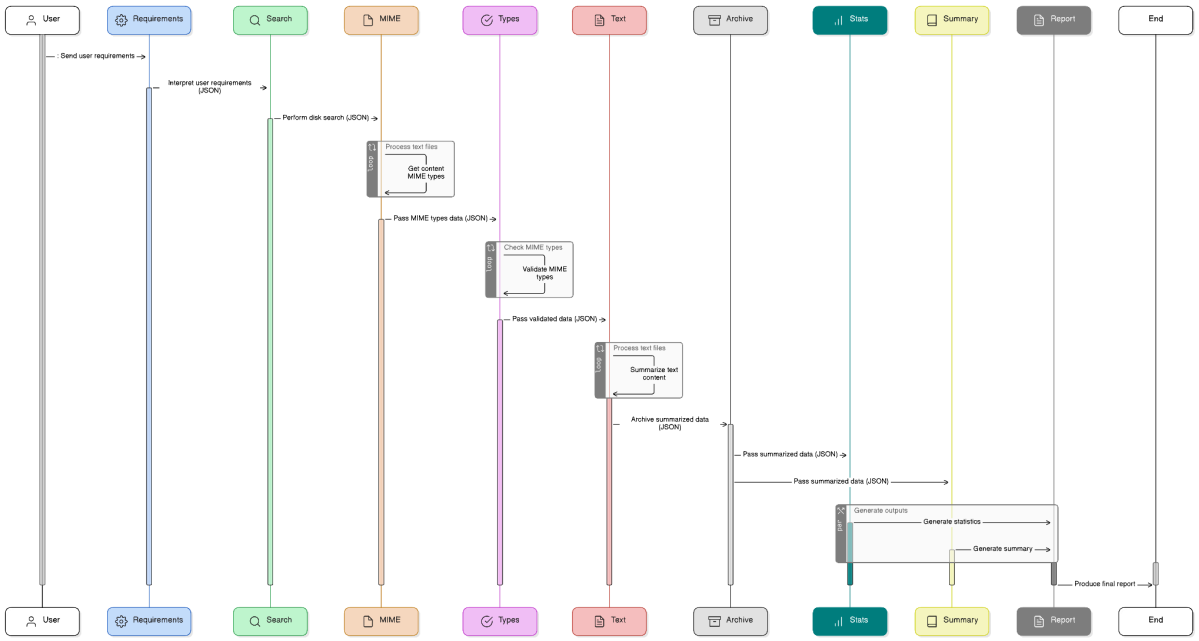

Fig 3: UML sequence diagram of the Digital forensics system

Challenges and key findings

Multi-agent systems have received tremendous attention as solutions for complex problems. Common challenges are task allocation and coordination between agents (Dorri, A., Kanhere, S.S. and Jurdak, R., 2018). The proposed approach uses a modular design which defines each agent's role and how it communicates to the next agent, enabling better organisation and maintenance. With agents operating independently in sequence, large datasets could introduce latency as the system waits for agents to complete their tasks before passing data on. Setting limitations to data size for example could solve this.

The Text agent that loops through the text files to produce one sentence summaries, may have severe problems with accurately summarising large text documents with only one sentence. There might be one massive text document in the library that delays the entire process.

Multi-agent systems have become known for their robustness and reliability; However, a common hurdle is that the agents often lack specific domain knowledge (Ren, Z. and Anumba, C.J., 2004). Since this design assigns agents simple tasks and gives them the relevant tools, it will not suffer from the main hurdle found in complex multi-agent systems. The simple sequential design also allows for easy testing and debugging, as it should be clear which agent is causing any bottlenecks. The design is scalable as agents can always be added or tweaked to improve performance or add more functionality to the system.

The main issue of this system is the expected performance overhead of running multiple agents. Agents will often have to wait for other agents to transmit data before they start their tasks. This will introduce potential bottlenecks and latency to the system. Dependencies on external libraries introduce risks related to making sure each agent is using the optimal library for their respective task and requires constant maintenance. Some libraries may become outdated or dysfunctional in future updates. The "spaCy" library that is used in this design was tested to be the most performance efficient library when compared to three other major libraries, though it still had issues with accuracy (Al Omran, F.N.A. and Treude, C., 2017).

Conclusion

This multi-agent based digital forensics system will allow users to quickly scan through folders to get a simple summary of what is within them. The proposed methodology explains the sequence of what tasks will be performed by which agents, using the specific tools they are given. Python gives access to an extensive list of libraries which should allow for plenty of solid options on which libraries each agent can possibly use to complete their respective task. This will require extra maintenance in managing which libraries each agent uses but will also increase the

system's longevity as there will likely be many good alternatives when a particular library becomes outdated.

Going forward it will be crucial to test how long each agent takes to complete their given tasks, to see where the potential bottlenecks may occur. Optimising each individual agent's ability to tackle their respective tasks is going to be critical to the success of the entire system. By addressing these concerns, the proposed design should provide a reliable and accurate digital forensics tool.

References:

Al_Kholy, M.S., Khalifa, A.R. & Alsaied, M.G. (2010) A Systematic Approach for Mobile Agent Design Based on UML. *Journal of American Science* 6(12): 284-290. Available from https://www.jofamericanscience.org/journals/am-sci/am0612/32_3283am0612_284_290.pdf [Accessed 30 August 2024].

Al Omran, F.N.A. & Treude, C. (2017) Choosing an NLP library for analysing software documentation: a systematic literature review and a series of experiments. *IEEE/ACM 14th international conference on mining software repositories (MSR)* 187-197.

Bauer, B., Odell, J. (2005) UML 2.0 and agents: how to build agent-based systems with the new UML standard. *Engineering Applications of Artificial Intelligence* 18(2005): 141–157. Available from https://www.sciencedirect.com/science/article/pii/S0952197604001903 [Accessed 30 August 2024].

Carvey, H. & Altheide, C. (2011) *Digital Forensics with Open Source Tools*. Amsterdam: Elsevier.

Dorri, A., Kanhere, S.S. & Jurdak, R. (2018) Multi-agent systems: A survey. *IEEE Access* 6: 28573-28593.

Ghazinour, K. et al (2017) 'A Study on Digital Forensic Tools', *IEEE International Conference on Power, Control, Signals and Instrumentation Engineering (ICPCSI)*. 21-22 September. DOI: https://doi.org/10.1109/ICPCSI.2017.8392304

Huget, M-P., Odell, J., Bauer, B. (2006) UML and Agents: Current Trends and Future Directions. *OOPSLA Workshop on Agent-Oriented Methodologies, Seattle, WA, USA, 4-8 November 2002*. Available from https://www.cin.ufpe.br/~in1096/2006-1/UML%20and%20Agents-%20Current%20Trends%20and%20Future%20Directions%20(Huget_uml)%5BB-I%5D.pdf [Accessed 30 August 2024].

Labrou, Y. & Finin, T. (1994) A semantics approach for KQML—a general purpose communication language for software agents. *CIKM '94: Proceedings of the third international conference on Information and knowledge management*: 447-455. DOI: https://doi.org/10.1145/191246.191320

Ren, Z. & Anumba, C.J. (2004) Multi-agent systems in construction–state of the art and prospects. *Automation in Construction* 13(3): 421-434.

Singh, G., Sharma, M., Singh, A. (2012) Using Intelligent Agents for Building Evacuation - A UML Approach. *International Journal of Computer Applications* 50(18): 14-17. Available from https://www.ijcaonline.org/archives/volume50/number18/7870-1147/ [Accessed 30 August 2024].