AI-assisted message processing for the Netherlands National Police

Bas Testerink
Police Lab AI
Netherlands National Police
Driebergen, The Netherlands
bas.testerink@politie.nl

Daphne Odekerken
Police Lab AI
Netherlands National Police
Driebergen, The Netherlands
daphne.odekerken@politie.nl

Floris Bex
Police Lab AI
Utrecht University
Utrecht, The Netherlands
f.j.bex@uu.nl

ABSTRACT

The number of messages that the Netherlands National Police (NNP) receives (e.g. from international partner institutes and citizens) grows steadily every year. The NNP has initiated a number of projects to develop artificial intelligence systems that assist in the processing of such messages. In this demo, we show a prototype of one such system that will be used for supporting the processing of messages from international (Interpol) partners.

1 INTRODUCTION

The number of messages that the Netherlands National Police (NNP) receives grows steadily every year. Such messages range from notifications from citizens to requests for assistance from international partner institutes. The NNP has initiated a project to develop artificial intelligence (AI) that assists in the processing of such messages, creating autonomous software agents that support human operators. The use of natural language processing tools is a cornerstone of the agents, because incoming messages are typically free-text (e-mails, online forms). Furthermore, it is important that the agents are designed in such a way that every major decision is made transparently, and that legal and ethical rules and regulations can be enforced.

The demo system enhances the existing processing of messages that are received through the Interpol channel. An overview of the goal system incorporated into the Interpol message processing workflow is shown in Figure 1. The pink components with human icons are the human operators. The orange components with the computer chips represent agent components. The yellow components are components without agency.

Currently, a *coordinator* monitors all the incoming messages and categorizes them on priority, theme and relevancy for The Netherlands. The coordinator may answer the message directly, forward the message to a *specialist* for further processing, or choose to ignore the message, for example because it is not relevant for the Netherlands. Specialists specialize in topics such as counter-terrorism and child sex tourism. Usually they have access to domain data and contacts that are relevant for their expertise. A specialist can forward a message internally or answer it directly.

In: Proceedings of the Workshop on Artificial Intelligence and the Administrative State (AIAS 2019), June 17, 2019, Montreal, QC, Canada.

The first agent that is inserted into the workflow is the *Triage Agent*, which supports the coordinator by performing classification and information extraction tasks. What is the theme, priority and relevance of the messages? Which entities (persons, bank accounts, countries, organisations) are mentioned in the message? What is the intent of the sender – are they asking for information, or do they expect the Dutch police to take action? Not all these questions can be answered given just the message. For instance, the occurrence of a person in the police databases may determine the relevancy for The Netherlands. The Triage Agent thus reads the e-mails and supports the coordinator. If the coordinator agrees with the agent, they forward the messages with the relevant annotations (entities, intent, priority level, etc.).

The second type of agent is a *Specialist Agent*, which supports specialists to do routine work on their respective themes. The agents that work for the specialists will formulate the task that is required given the message, execute it, and then report their findings to the specialist as an enhancement of the initial message. The idea is that the specialist ultimately receives messages as if a colleague already processed it. For instance, consider a notification that during a routine border patrol a Dutch vehicle was found to contain illegal drugs. The Triage Agent already determined the vehicle to be indeed Dutch and the message is forwarded to the specialist agent for drug related crime. This specialist agent has access to current drug-related investigations such as which organizations are of special interest. It tries to match the notification to existing investigations or otherwise initiates a new one. By the time the specialist receives the notification from agent he or she can immediately see how the notification relates to past information and what course of action would be prudent. The main task of the specialist then becomes the monitoring and training of the agent.

A final piece of functionality will be to aggregate the messages. At the NNP we are working on real-time monitoring of intelligence data from international partners, combining it with open source intelligence (news, Wikipedia, Twitter) and in-house intelligence from the NNP.

2 AGENT ARCHITECTURE

The architecture we use for the individual (Triage and Specialist) agents is the same architecture that we have used in our other project *Intelligence Amplification for Cybercrime* (IAC) [1], in which we have designed an agent to assist the NNP in the assessment of crime reports submitted by civilians. In a nutshell, the agent applies information extraction techniques to understand a document, applies legal reasoning to determine whether more information

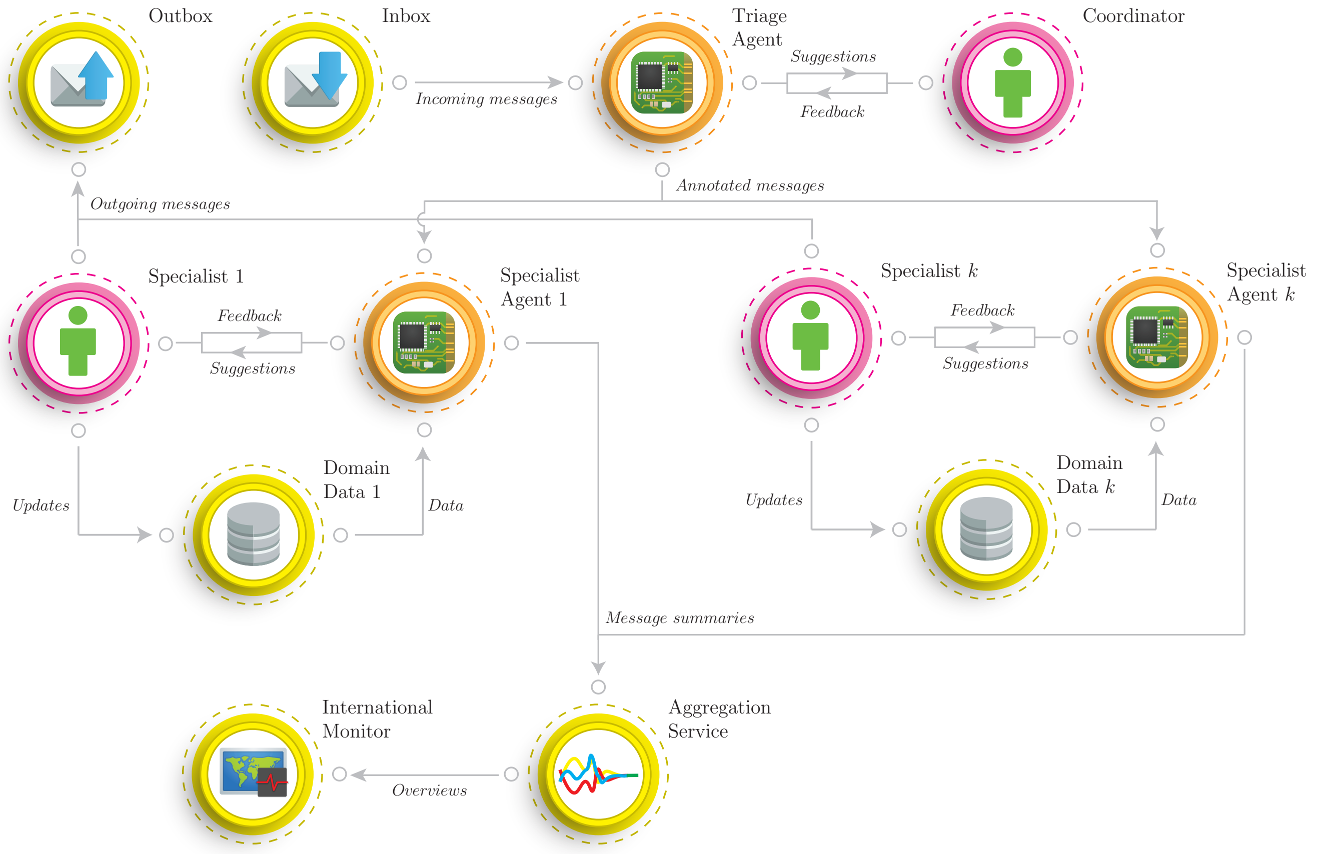

Figure 1: AI-assisted message processing multi-agent system overview.

is required and applies a policy that is optimized for efficiency to determine the next action.

The agent's goal is to produce some (information) product such as a report, reply or analysis. We refer to a coherent sequence of interactions with the environment as a session. For example, a session can be a sequence of database queries that were required to respond to a message. A session always ends with a terminal action. For instance, terminal actions can be to ignore the message, forward it or answer it directly.

For many law-enforcement applications we need to be able to check why the agent suggests (or directly executes) some terminal action. The basis for many decisions is legislation. Hence, we draw upon the field of computational legal argumentation (cf. [2]) to ensure that the agent has an argument grounded in the relevant rules and regulations when it decides upon a terminal action. The agent architecture is designed for creating agents that efficiently seek information in their environment and transparently decide on what terminal action ought to be executed [4].

Figure 2 shows an overview of the agent architecture. The deployment phase concerns the actual functionality of the agent. The training phase is required to configure the deployment phase components. The deployment components are the top-half (blue) components. The training components are the bottom-half (green) components. The monitor interface and argumentation engine are used in both phases.

2.1 Deployment

The agent connects to its environment through an *external interface*. That interface differs per application. In the message processing system, it will contain functionalities such as forwarding e-mails and querying databases. Typically, the external interface is implemented as a layer that calls different APIs of other systems and passes on the callback. The aim of the agent is to create some (information) product such as a message which is annotated with analyses and suggested actions. These products are stored in the *product database*. Such a product is typically built in two phases: first the agent tries to find enough information to make a final decision on the product, and second the final decision results in a terminal action.

External information, such as a message and database results, are feedback which the external interface sends to the internals of the agent. The feedback is put through a pipeline of *classifiers and attribute extractors* which turn the feedback into structured data (statements about the feedback which are attributes and Boolean observations). For our earlier IAC intake agent, we use existing named entity recognition software [3] and bespoke classifiers (cf. Section 2.2) to classify and extract entities (e.g. the suspect, the

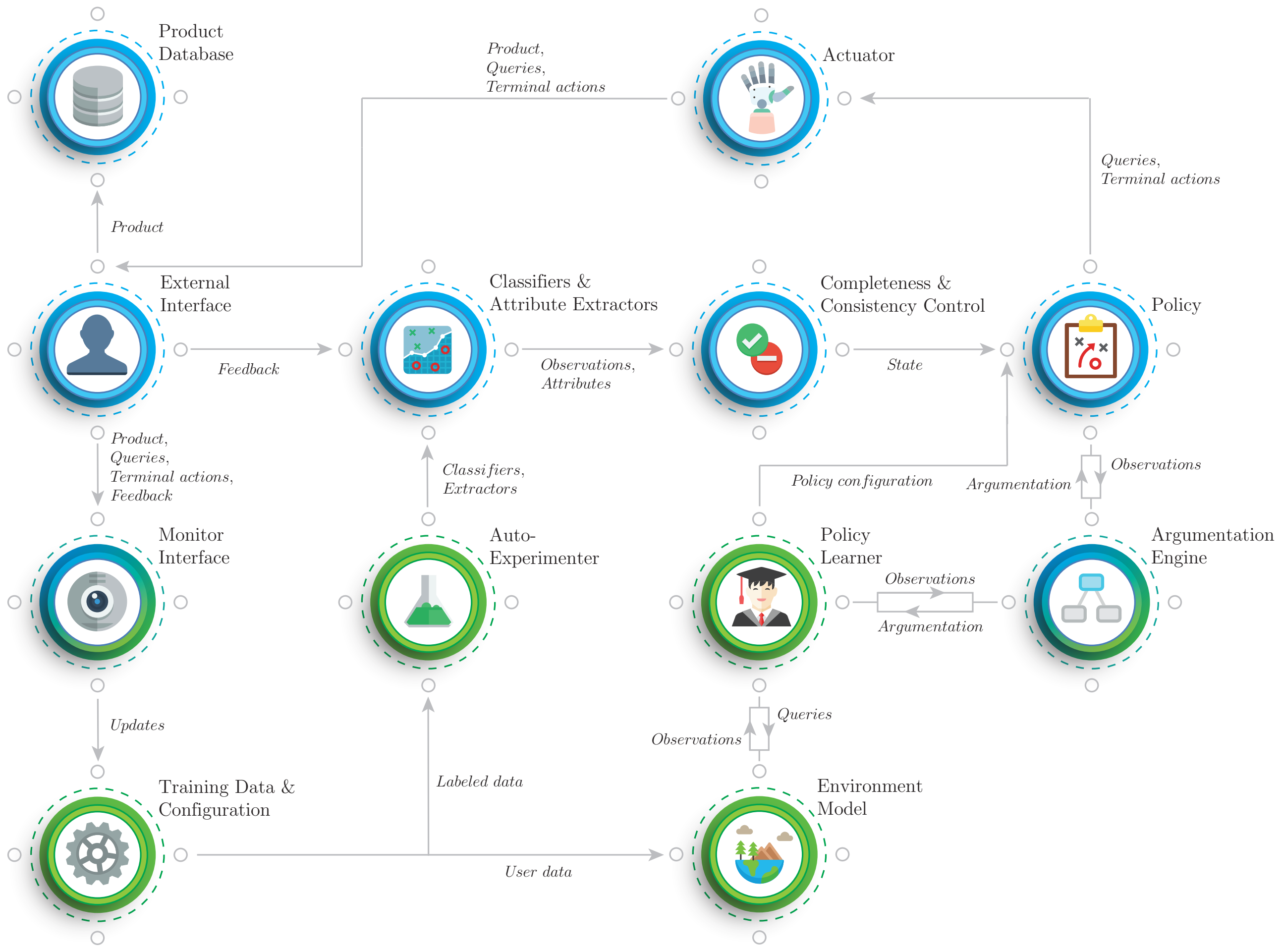

Figure 2: Agent architecture.

victim, addresses) and relations (e.g. “the suspect received money from the victim”). For the Interpol Triage agent we use Spacy as the basis and apply pre- and post-processing to improve upon its base performance. Our choice for Spacy was based on its ease-of-use and available multi-language models for NLP.

The classification and extraction pipeline consists of many separately constructed components which may result in inconsistent results. Hence, we apply a *consistency control* mechanism which makes sure that the data is consistent. It also checks whether the data is complete. That check is mainly for fulfilling the preconditions of final actions, e.g., administrative requirements. The result of this controller is what we consider to be the state of current session, where a session is a sequence of actions after the initial input until the information product is produced (i.e. the suggested course of action for the user).

The actual decision making of the agent is executed by a decision-making *policy*. Based on the state, it determines the next action; this can be an information gathering action (query) or a terminal action. The policy may draw upon the *argumentation engine* in order to argue for or against an action. The *actuator* of the agent prepares the action for execution through the agent’s external interface. For instance, the action might be an information gathering action upon a database. The actuator may then formulate an SQL query which the external interface ensures is sent to the appropriate database and returns the result as feedback to the classification and attribute extraction pipeline.

2.2 Training

The *monitor interface* allows a human operator to monitor the agent’s activities and control its training phase. The human operator uses the monitor interface also to create labelled data by approving or disapproving (part of) the agent’s activities. The monitor interface also shows the argumentation behind core decisions. This connection is not shown in the figure as showing the argumentation does not directly impact the agent’s decision-making. However, it does help the human operator to understand the choices of the agent and localize where potential corrections have to be made. The training of the agent is based on *example data and configuration settings* of its different training tasks. For the observations and attributes, we apply supervised learning which is enabled by the gathering of labelled data during deployment. An *automated* *experimenter* module tries different algorithms in order to determine for each observation and attribute what the best classifier or extractor is.

The policy of the agent is shaped by reinforcement learning (although other methods can be used). The *policy learner* tries to create a policy which efficiently interacts with the environment. For instance, it may try to minimize the amount of data that it queried from databases. In order to practice these interactions, the policy learner requires an *environment model* that is generated from the training data. The model captures for instance probability distributions over random variables that the agent encounters. At the moment the model's implementation is a Bayesian network where its nodes are observations that the argumentation module may use to construct arguments with. For reinforcement learning, we apply the argumentation engine as part of its reward function. The agent gets a positive reward when it achieves a state such that more feedback from the external interface cannot change its opinion on the terminal action. When such a state is reached, it is natural to opt for the terminal action that the agent can argue for [4].

3 DESIGN CONSIDERATIONS

The application of autonomous A.I. systems requires careful considerations with respect to their potential impact. We decided to restrict the agent's capabilities to reading messages, querying systems and presenting information to the human operator since we cannot guarantee correct behaviour due to the agent's reliance on imperfect information extraction. The agent has no capability for updating databases or sending messages without explicit approval from the human operator.

During deployment, every decision outcome of the agent is documented in its trace for *auditability* and can be inspected by a human operator. However, it should be noted that it is not always possible to completely reproduce the behaviour of the agent: some queried databases contain data that is forbidden by law to store in the same environment. Hence, it is for instance not allowed to store raw database query results. As a result, there are situations in which it cannot be reproduced which information the agent exactly had when it made a decision. This happens for example when source databases are updated. The human operator can provide feedback through the monitor interface which can be taken into consideration when the system is retrained/adjusted.

Interpretability has been an import design influence from the start. It was determined early on that extracting information from data will be a hard to interpret exercise under most circumstances. From this point of view, it was not desirable to design the application as an end-to-end system. Instead, it was opted to create a method where the granularity of extraction can be balanced with interpretability and transparency. In short, we designed the system in such a way that we can choose how much information is obtained through extraction techniques and how much is inferred by argumentation. Generally the trade-off is accuracy vs. transparency.

In order to increase and maintain the *accuracy* of the agent, we rely on three pillars: A) human operators keep providing training data, which is done not only for keeping the data up-to-date, but also to comply with expiry dates of data; B) the auto-experimenter rigorously searches for the best models; and C) collaborations with academia ensure that the latest academic results are tried and tested.

4 CONCLUSION

In this paper we briefly discuss a multi-agent architecture for handling messages from international Interpol partners to the NNP, as well as the architecture of a single agent. In our live demo, we show the workings of a single Triage Agent with a realistic example. We encourage interested programmers to contact the authors for the source code.

REFERENCES

[1] F.J. Bex, J. Peters, and B. Testerink. 2016. AI for online criminal complaints: From natural dialogues to structured scenarios. In *Artificial Intelligence for Justice Workshop (ECAI 2016)*. 22–29.
[2] H. Prakken and G. Sartor. 1996. A dialectical model of assessing conflicting arguments in legal reasoning. In *Logical models of legal argumentation*. Springer, 175–211.
[3] M.P. Schraagen, M.J.S. Brinkhuis, and F.J. Bex. 2017. Evaluation of Named Entity Recognition in Dutch online criminal complaints. *Computational Linguistics in the Netherlands Journal* 7 (2017), 3–16.
[4] Marijn Schraagen, Bas Testerink, Daphne Odekerken, and Floris Bex. 2019. Argumentation-driven information extraction for online crime reports. In *First Workshop on Legal Data Analytics and Mining (LeDAM 2018)*.