AFRL-RH-WP-TR-2022-0105

# STRATEGIES FOR INVESTIGATING AND ELICITING INFORMATION FROM NUANCED ATTACKERS (SIENNA)


**Brian Krisler, Ph.D. / Aaron Reed / Adam Summerville**
**Zech Bertilson / Alexander Christner / Ian Cook**
**Nathan Lenz / Matthew Vincent / Dustin Wright / Katelyn Carino**
**James Ryan / Jordan Hashemi / Bill Ferguson**
**Raytheon Technologies Corporation**
**1000 Wilson Blvd.**
**Arlington, VA 22209**


## December 2022

## Final Report

**Distribution A. Approved for public release; distribution unlimited.**


**AIR FORCE RESEARCH LABORATORY**
**711TH HUMAN PERFORMANCE WING**
**AIRMAN SYSTEMS DIRECTORATE**
**WRIGHT-PATTERSON AIR FORCE BASE OH 45433**
**AIR FORCE MATERIEL COMMAND**
**UNITED STATES AIR FORCE**

## NOTICE AND SIGNATURE PAGE

Using Government drawings, specifications, or other data included in this document for any purpose other than Government procurement does not in any way obligate the U.S. Government. The fact that the Government formulated or supplied the drawings, specifications, or other data does not license the holder or any other person or corporation; or convey any rights or permission to manufacture, use, or sell any patented invention that may relate to them.

This report was cleared for public release by the AFRL Public Affairs Office and is available to the general public, including foreign nationals. Copies may be obtained from the Defense Technical Information Center (DTIC) (http://www.dtic.mil).

AFRL-RH-WP-TR-2022-0105 HAS BEEN REVIEWED AND IS APPROVED FOR PUBLICATION IN ACCORDANCE WITH ASSIGNED DISTRIBUTION STATEMENT.

ANDERSON.TIMOTHY.RAY.1230210728 Digitally signed by ANDERSON.TIMOTHY.RAY.1230210728 Date: 2023.03.23 07:53:09 -04'00'

TIMOTHY R. ANDERSON, DR-IV, Ph.D.
Work Unit Manager
Mission Analytics Branch
Airman Systems Directorate
711th Human Performance Wing
Air Force Research Laboratory

MURDOCK.WILLIAM.P.1048742161 Digitally signed by MURDOCK.WILLIAM.P.1048742161 Date: 2023.03.23 11:17:02 -04'00'

WILLIAM P. MURDOCK, DR-IV, Ph.D.
Chief, Mission Analytics Branch
Airman Systems Directorate
711th Human Performance Wing
Air Force Research Laboratory

CARTER.LOUISE.ANN.1230249128 Digitally signed by CARTER.LOUISE.ANN.1230249128 Date: 2023.03.23 14:59:01 -04'00'

LOUISE A. CARTER, DR-IV, Ph.D.
Chief, Warfighter Interactions and Readiness Division
Airman Systems Directorate
711th Human Performance Wing
Air Force Research Laboratory

This report is published in the interest of scientific and technical information exchange, and its publication does not constitute the Government's approval or disapproval of its ideas or findings.

# REPORT DOCUMENTATION PAGE

PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ORGANIZATION.

| 1. REPORT DATE | 2. REPORT TYPE | 3. DATES COVERED START DATE | END DATE |
|---|---|---|---|
| 12/15/2022 | Final | 28 September 2018 | 15 December 2022 |

**4. TITLE AND SUBTITLE**
Strategies for Investigating and Eliciting Information from Nuanced Attackers (SIENNA)

| 5a. CONTRACT NUMBER | 5b. GRANT NUMBER | 5c. PROGRAM ELEMENT NUMBER |
|---|---|---|
| FA8650-18-C-7885 | | |
| **5d. PROJECT NUMBER** | **5e. TASK NUMBER** | **5f. WORK UNIT NUMBER** |
| | | H0X1 |

**6. AUTHOR(S)**
Brian Krisler /Aaron Reed / Adam Summerville / Zech Bertilson / Alexander Christner / Ian Cook / Nathan Lenz / Matthew Vincent / Dustin Wright / Katelyn Carino / James Ryan / Jordan Hashemi / Bill Ferguson

| 7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) | 8. PERFORMING ORGANIZATION REPORT NUMBER |
|---|---|
| Raytheon BBN Technologies<br>10 Moulton Street<br>Cambridge, MA 02138 | |

| 9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) | 10. SPONSOR /MONITOR'S ACRONYM(S) | 11. SPONSOR/MONITOR'SREPORT NUMBER(S) |
|---|---|---|
| Air Force Research Laboratory<br>711th Human Performance Wing<br>Airman Systems Directorate<br>Warfighter Interactions and Readiness Division<br>Wright-Patterson Air Force Base, OH 45433 | | AFRL-RH-WP-TR-2022-0105 |

**12. DISTRIBUTION/AVAILABILITY STATEMENT**
Distribution A. Approved for public release; distribution unlimited.

**13. SUPPLEMENTARY NOTES**
AFRL-2023-1059; Cleared: 2 Feb 2023

**14. ABSTRACT**
This research was funded under the DARPA Active Social Engineering Defense (ASED) program. Our effort involved counter-engaging adversaries by gaining their trust and efficiently eliciting information from them, accomplished through our capability of socially coherent attacker investigations. SIENNA is a partial TA2 solution entailing the construction and deployment of a bot framework driven by conversational technology that members of our team originally devised in the context of videogames. When an attack is recognized, SIENNA will deploy a set of bots to engage and investigate the attackers. Each bot has a role, goals, and speaking style (its “persona”) selected by SIENNA to exploit what it knows so far about the nature and goals of each attacker. The bots’ true purpose is to engage, build trust, provide fake information, and most importantly to elicit information from the attacker and waste their time and resources. Additionally, SIENNA was awarded an Engineering Change Proposal to expand the scope of the Quixote quest module and Cervantes Web-based Quest Editor that utilizes a templated structure to develop detailed narrative Quests.

**15. SUBJECT TERMS**
ASED / SIENNA

| 16. SECURITY CLASSIFICATION OF: a. REPORT | b. ABSTRACT | c. THIS PAGE | 17. LIMITATION OF ABSTRACT | 18. NUMBER OF PAGES |
|---|---|---|---|---|
| U | U | U | SAR | 81 |

| 19a. NAME OF RESPONSIBLE PERSON | 19b. PHONE NUMBER *(Include area code)* |
|---|---|
| Timothy R. Anderson, Ph.D. | |

# TABLE OF CONTENTS

# LIST OF FIGURES



# LIST OF TABLES

# 1.0 SUMMARY

This report details a novel approach of using chatbot technologies for engaging with interlocutors, while actively soliciting information through the use of distinct virtual personas realized in software agents. Titled Strategies for Investigating and Eliciting Information from Nuanced Attackers (SIENNA), this research was conducted under Technical Area (TA) 2 of the Defense Advanced Research Project Administration's (DARPA) Active Social Engineering Defense (ASED) program.

The product of our research consists of two primary technologies:

- SIENNA-Bot: A chatbot designed to converse with an interlocutor using domain-specific content.
- Cervantes: A graphical user interface (GUI) for domain-specific dialogue development that evolves around the concept of *quests*, i.e., series of questions of increasing complexity intended to elicit information from the interlocutor.

Critical to the success of TA2 is the generation of logical and coherent dialogue. This dialogue should be effective in engaging and interacting with an interlocutor as if they are communicating with another human. To generate content, we adopted a novel authoring scheme for natural language generation that is driven by attribute grammars. The SIENNA-Bot follows a pipeline design approach of:

- *Natural Language Understanding (NLU)*. The intent of the incoming message is ascertained and assigned attributes.
- *Dialogue Generation*. The ascertained content attributes are evaluated against the conversational state to determine the next moves.
- *Natural Language Generation (NLG)*. A viable response to the message is generated.

To demonstrate the effectiveness of this design prior to investing time towards designing and developing the language understanding and generation components, we developed, we developed an initial proof of concept chatbot with a simple dialogue generator. The goal of this bot was to confirm the efficacy of engaging an interlocutor in conversation through simple discourse techniques without having to develop the necessary components for parsing and comprehending message content. The resulting bot was capable of countering impersonation attacks by taking on one of two distinct personas:

- *Needy narcissist*. Loves to gossip about coworkers and colleagues in the targeted field, expresses distaste for recent colleague successes and likes to tell rambling stories about themselves.
- *Spiteful colleague*. Leverages a narrative conceit to introduce a fictional backstory as a means of conversation. Because the interlocutor only pretends to know the subject:
    - A fictional story allows the bot to quickly take control of the conversation

- No comprehension of the message content is required

Building upon this foundational bot, we next investigated techniques for introducing realism into the conversation by incorporating NLU and NLG.

The objective of NLU modules within SIENNA are to:

- Understand the pragmatics of what the interlocutor is saying
- Extract critical pieces of information from the interlocutor's messages

This objective was accomplished using a pre-trained Transformer model with added layers for understaning the pragmatics of the interlocutor's message while simultaneously extracting critical pieces of information, called *flags*. The resulting NLU was capable of classifying various types of discourse-acts with the key benefits providing:

- Dialogue state information
- Affordances for the content authoring

For example, by detecting that an interlocutor is arguing when responding to a quest, SIENNA could use that information to either change the direction of the quest, or assign a new, potentially easier quest. The concept of trust tracking was also incorporated into SIENNA's NLU with the guiding principles being:

- Difficult requests require high levels of trust
- Trust rises when quests are completed

Working from these axioms, we designed NLU functionality to combine the successful completion of a quest with the determined compliance of the content of each message. If a quest went uncompleted, or the NLU module detected affects such as *anger, frustration,* or *impatience* in the messages, overall trust was decreased. Conversely, if the interlocutor was compliant and willing to answer questions, the level of trust went up.

We further evolved the Dialogue Generation functionality by incorporating two techniques:

- *Conversational state*. Maintaining and using prior conversation content such as
    - *Preconditions* to determine if dialogue content should or should not be used
    - *Effects* to establish and maintain state for continued future dialogue
- *Custom Dialogue Generators*. These dialogue generators provided specialized content generation designed to handle specific details of a conversation

This work resulted in an NLG module capable of producing dialogue based on the semantics of the input message through the process of *quibbling*, arguing and raising objections to a trivial matter, with the interlocutor.

With the addition of these new techniques, SIENNA bots incorporated a level of conversation understanding and consistency well beyond the initial “dumb” bot strategy planned at project inception.

The procedural content generation approach taken by SIENNA requires an ability to create the conversational domain. Specifically, having the functionality for non-developers and non-SIENNA experts to construct new quests. SIENNA uses a strategy of asking the interlocutor questions of increasing complexity, to consume the time of an interlocutor while simultaneously acquiring information from them. These questions are called quests. In order to enable a content creator to author a series of quests for a domain, we developed a user-friendly editor called *Cervantes*.

The first step in the creation of the Cervantes editor was the design and development of a domain-specific language (DSL). By using a DSL, we established a formal structure that promoted quick iterations on the Cervantes design. The semantics of the DSL include:

- Definition Blocks. These blocks set up reusable moves, responses, and behaviors across a whole set of quests.
- Quest Blocks. A particular “mission” that the bot attempts to get the interlocutors to waste time on and/or reveal information while performing.
- Conditions and Effects. Provide the author with the affordances to link together series of quests and allow the SIENNA quest manager to transition between quests dynamically, as the conversation evolves.

The resulting feature set of Cervantes includes:

- Detailed Quest Editor: Ability to create tokens, variables and conversational dialogue
- Embedded Help: The user interface (UI) has a hyperlinked, embedded help system
- Quest Simulator: A simulator for testing and debugging quests during development
- Multi-User Interface: Functionality for supporting multi-users
- Rights and Roles: Rights and roles modelled from common open-source platforms, such as GitHub, where users are members of projects, and each project has specific rights and roles
- Versioning: The ability to create project versions from within the UI

Over the course of the program, we developed a quest library of 50 quests. Furthermore, we deployed Cervantes on a shared server, accessible by all performers with multiple non-Raytheon BBN Technologies (BBN) collaborator teams creating quest libraries such as the Corona Virus Disease (COVID) quest library and the Court Summons Library.

In summary, the SIENNA approach of using expressive chatbots to engage with interlocutors demonstrated effective and validating results. Over the course of the program, the number of true positive flags captured by SIENNA increased from 71% to 89%. Multiple non-SIENNA team organizations were successful at using Cervantes to create their own domain-specific quest libraries.

Our principal recommendations for the further development of SIENNA are:

- Multi-Lingual Support.
- Human-in-the-Loop.
- Group Bot Interaction.
- Leverage External Information When Available.
- Improved cross-platform switching.

# 2.0 INTRODUCTION

BBNs SIENNA project under the DARPA ASED program involves counter-engaging adversaries by gaining their trust and efficiently eliciting information from them, accomplished through our capability of socially coherent attacker investigations. SIENNA is a partial TA2 solution entailing the construction and deployment of a bot framework driven by conversational technology that members of our team originally devised in the context of videogames and significantly expanded under SIENNA. When an attack is recognized, SIENNA deploys a set of bots to engage and investigate the interlocutors. Each bot has a role, goals, and speaking style (its "persona") selected by SIENNA to exploit what it knows so far about the nature and goals of each interlocutor. The bots' true purpose is to engage, build trust, provide fake information, and most importantly to elicit information from the interlocutor and waste their time and resources. To construct and operate our bots, we leveraged existing, innovative persona-authoring and history-generating tools. These tools make SIENNA both highly customizable and scalable.

The BBN team approached this program with two key hypotheses:

1. By keeping an interlocutor engaged, we can occupy their time enough that they will not have time to phish other individuals.
2. As trust increases between the bot and the interlocutor, the complexity of the quests can increase.

To validate these hypotheses, BBN pulled together a team of experts. With the help of the University of Maryland, our team extended an existing experimental platform to discover linguistic techniques for building trust and exchanging resources, which we then used in the construction of our bots. Dr. Adam Summerville at Cal Poly Pomona provided the machine learning (ML) expertise to allow our team to design and build bots capable of generating coherent content to augment and supplement the generative human authored text.

Additionally, SIENNA was awarded an Engineering Change Proposal to expand the scope of the Quixote quest manager module and Cervantes Web-based Quest Editor that utilizes a templated structure to develop detailed narrative Quests. In collaboration with teammates and subject matter expert Dr. Aaron Reed, we developed both a domain-specific language and user interface for capturing and compiling human authored content to be used by the SIENNA bot to strategically deploy domain-specific content.

To evaluate our conversation bots, SIENNA participated in multiple government-sponsored experiments. In addition to those experiments, we stood up a government-sponsored Cervantes (the BBN developed authoring tool) instance and provided access to all the other program performers to construct their own dialogs and provide feedback. The results of both the bot evaluations and the user interface work will be provided below.

The following sections in the report will detail both the approach taken and results achieved in our work.

## 3.0 METHODS, ASSUMPTIONS, AND PROCEDURES

### 3.1 Assumptions

SIENNA was developed under TA2 of the DARPA ASED program. As part of our research, assumptions were made as to the program structure and the technical goals.

The following infrastructure assumptions were made during our research.

- It was assumed that other Technical Area performers had successfully identified and marked all incoming phishing messages. It was not the objective of SIENNA to determine if a message was or was not legitimate. Given this assumption, SIENNA was able to immediately react to all received messages, extract appropriate flags and craft a valid response.
- It was also assumed that other Technical Area performers set up the mail infrastructure to send and receive emails. SIENNA assumed all interactions would occur via an intermediary through RESTful Application Programing Interface (API) calls. Given this assumption, we built SIENNA with a representational state transfer (RESTful) interface and added message metadata indicating when a message should be delivered. It was then the goal of the other components to handle the scheduling and delivery of messages, as an assurance that both new messages and responses made their way into SIENNA.
- SIENNA was built and delivered as a container-based application using a modern Continuous Integration, Continuous Development (CI/CD) pipeline. It was assumed that a cloud-based infrastructure existed to pull and deploy new versions of SIENNA as they were developed, and that the infrastructure existed to ensure the proper and successful linking of SIENNA to the other performers components.
- SIENNA and Cervantes were designed to operate independently of one another. Given this, it was assumed that procedures existed to move newly authored quests from the Cervantes system to the system running the SIENNA bot. This assumption was intentional to allow content authors the ability to freely author new content on different networks from the bots.

The following technical assumptions were made to validate our hypotheses:

- Using a more generative language model (LM) for dialogue construction, the resulting bots would express a more humanlike dialogue allowing for more engagement and flag captures.
- Designing a bot architecture that allows for pluggable expertise, such as specialized language generation models, will promote a more sophisticated dialogue.
- By establishing a formal domain language for initial dialogue language, user interfaces can be quickly developed to support non-programmer dialogue construction.

- Using the concept of quests provides a higher-level design principle that will allow for increased interactions with an interlocutor as well as increased flag captures.
- Continuously capturing and monitoring trust will provide mechanisms for determining when to increase or decrease quest difficulty.

## 3.2 Methods and Procedures

To test and validate our technical assumptions, the following methods and procedures were used.

### 3.2.1 Software and Development Methods

SIENNA followed an agile development approach and used a GitLab CI/CD pipeline to build new service containers. SIENNA followed the standard GitLab release process as documented here: https://docs.gitlab.com/ee/user/project/releases/#create-a-release.

The SIENNA release process had the following roles and responsibilities to ensure a consistent process without compromising the larger continuously operating system.

#### 3.2.1.1 Release Manager

- Create a milestone for the new version
- Assign issues to the milestone
- When all issues in the milestone are closed, a release is created for the version
- Add all issues to the release notes

#### 3.2.1.2 Developer

- Create an issue for work performed
- Close issues when finished
- If an issue is closed that was not assigned to a milestone, assign it to the next release version milestone

Versions followed a semantic versioning format of: **major.minor.patch** with most releases only changing the minor version.

- **Major:** Major breaking changes are introduced. Large new features introduced.

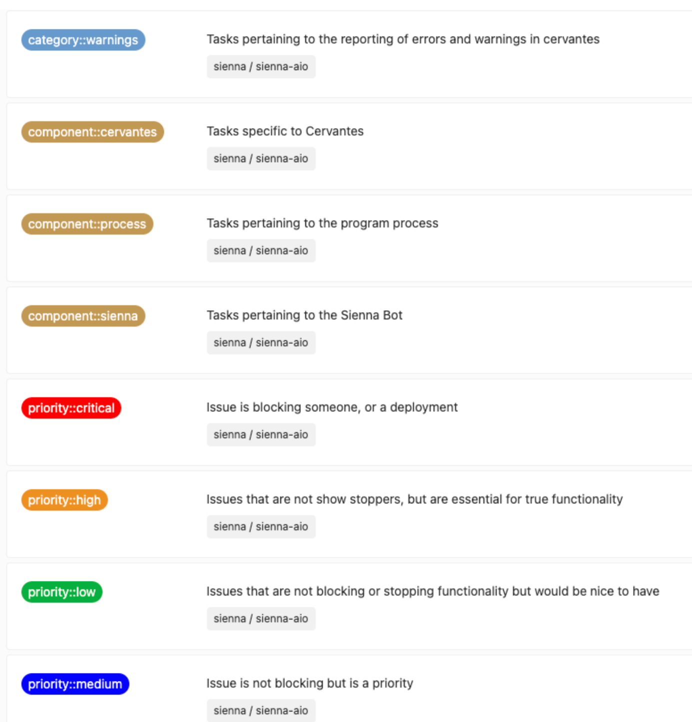


**Figure 1. GitLab Labels were Used to Categorize and Prioritize Issues.**

- **Minor:** Breaking changes are unlikely. New features introduced.
- **Patch:** No breaking changes. Usually used for bug fixes.

An issue tracker was used across all program performers for reporting and monitoring issue fixes. **Milestones** were used to mark release points and **labels** (See Figure 1) were used to categorize and prioritize issues as they were recorded.

The SIENNA build process occurred over three stages:

- **Pre-Check:** Performs a few pre-checks to ensure build will proceed successfully.
- **Build:** Builds all of the SIENNA components, including compiling and generating the help documents.
- **Scans:** Performs software scans such as linting.

### 3.2.2 Technical Methods and Procedures

The SIENNA approach to conversation dialogue was designed around two primary concerns (See Figure 2), Authoring and Execution.

Authoring focused on creating the tools for subject matter experts to easily create and modify new dialogue content through the form of quests. Execution dealt with the actual usage of that generated dialogue during active engagements with an interlocutor.

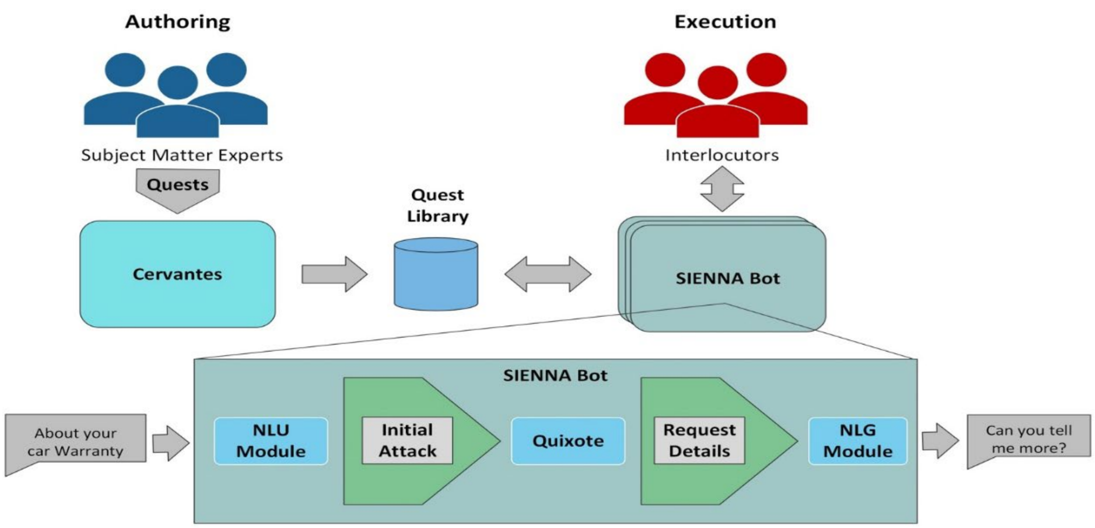


**Figure 2. High-Level Representation of the Technical Approach and Methods used for SIENNA.**

### 3.2.3 Authoring

With SIENNA, we set out to build a system that could structure interactions between a human interlocutor and a bot around the concept of "quests," and captured key-value pairs of personally identifying information about interlocutors, called "flags." To achieve this goal, we constructed a dynamic quest generation system that takes into account the level of trust we detect in the interlocutor with the SIENNA bot. The dialogue generator starts with easy quests and increases the difficulty as a way to slowly reinforce interlocutor engagement, backing off temporarily if the interlocutor starts to seem suspicious. This technique was chosen to maximize interlocutor engagement compared to a system that simply assigns quests randomly. In order to get relevant content for quests, and to ensure they do increase in difficulty, our system was designed around the inclusion of human quest authors. On that end, we developed authoring tools to ease the creation, modification and deployment of new dialogue into the system. The following sections will document the procedures taken to design and develop these components and their underlying subcomponents.

#### 3.2.4 CervantesDSL

To establish a well-defined content framework for ensuring the stability and consistency of Cervantes as an authoring tool, we designed a DSL. DSLs are specialized languages crafted to address the needs and requirements of a specific domain. Using the DSL, domain experts can author new SIENNA projects by hand using a procedural narrative in any text editor. The following sections provide an overview of the CervantesDSL. See Appendix B - CervantesDSL for the complete CervantesDSL specification.

CervantesDSL files are text-based files identified with a *.cervantes* extension. Each Cervantes file consists of a series of blocks, which may be defined in any order. There are two kinds of blocks that make up a Cervantes file: definition blocks and quest blocks. These blocks, as seen in Figure 3, are written in text using a well-defined format that is parsed by a SIENNA developed compiler.

```
Quest "Get Phone Number"
"""Ask for the attacker's phone number and capture it."""
Easy
Captures PHONE_NUMBER_PRIMARY
* Assign when {flag.PHONE_NUMBER_PRIMARY does not exist}
        "Hey, [[could_you]] send me a phone number I can reach you at???"
        #then {track.some = "variable"}
* Reassign
        "Still waiting on that phone number, [[when_youre_free]]."
* Negotiation
        "I'm not going to negotiate with you about a phone number."
* Giveup
        "You know what, never mind. I'm happy to keep using email."
* Question
        "[[Sorry,]] I'd just like to have another way to reach you in case something important
comes up... email can be really flaky[[maybe_emoji_endpunc]]"
* Finish when {flag.PHONE_NUMBER_PRIMARY exists}
        "Good to have a backup way to reach you! [[Thanks.]]"
* Finish when {temp.attacker_says_no_phone_number exists}
        "That's fine, I'll just use your email."
```

**Figure 3. Example CervantesDSL Quest Block.**

#### 3.2.4.1 Definition Blocks

Definition blocks set up reusable moves, responses, and behaviors across a whole set of quests, while each quest is a particular "mission" that the bot uses to attempt to get the interlocutor to waste time or reveal information while performing.

#### 3.2.4.2 Quest Blocks

A Quest represents a single topic from the bot, and perhaps some number of back-and-forth interactions discussing that request, ending with the interlocutor ultimately either fulfilling the request (i.e., by supplying some requested info) or failing to. When the interaction moves on to a different topic, that means transitioning to a new quest (even if a series of quests are linked together or conceptually related).

#### 3.2.4.3 DSL Compiler

Once a DSL project has been authored, it must be compiled to runtime matter usable by the bot. To address this need, we developed a compiler for the CervantesDSL. The compiler turns a *.cervantes* source file and included external libraries into a set of runtime files describing a SIENNA conversational domain.

A *Conversational Domain* is a set of files that lets SIENNA generate responses to incoming messages. It represents a particular scenario for which content has been created and a strategy for the SIENNA-Bot to respond to interactions with an interlocutor. Cervantes is a tool for easily defining a conversational domain.

The compiler generates the following files:

- conversational_domain_definitions.py: This file configures the SIENNA bot for the given conversational domain, and is read in by SIENNA at startup.
- quest_definition.json: Defines all the quest content for this scenario and related materials created in the authoring tool. Read in by the quest manager module (Quixote).
- state.schema: Defines all variables that might be referenced in this scenario, both custom and imported. Used by the compiler for validation; not currently imported at runtime but present if needed for reference.
- {project}.json, .marisa, .meanings, .stats, .grammar: This set of five files is used by the SIENNA bot to efficiently generate runtime text. All "tokens" defined or imported in a project are represented here.
- compiled_source.cervantes: The CervantesDSL code which the compiler uses to generate the other files. In most cases, this will have been generated by the Cervantes GUI. This is not needed at runtime but is presented for reference to aid in debugging.

### 3.2.5 The Cervantes Tool

We used the CervantesDSL as the basis for designing a GUI to ease the creation of quests. We designed this interface, called Cervantes, with the intent of having an easy way for non-developers to create quests without having to understand complex formal language semantics.

The procedure for designing Cervantes was to identify all the high-level concepts in the DSL and then create user friendly editors for each concept. Figure 4 illustrates the resulting Cervantes functional map. We broke functionality down into two categories: quest authoring and quest testing and evaluation, described in the following sections.

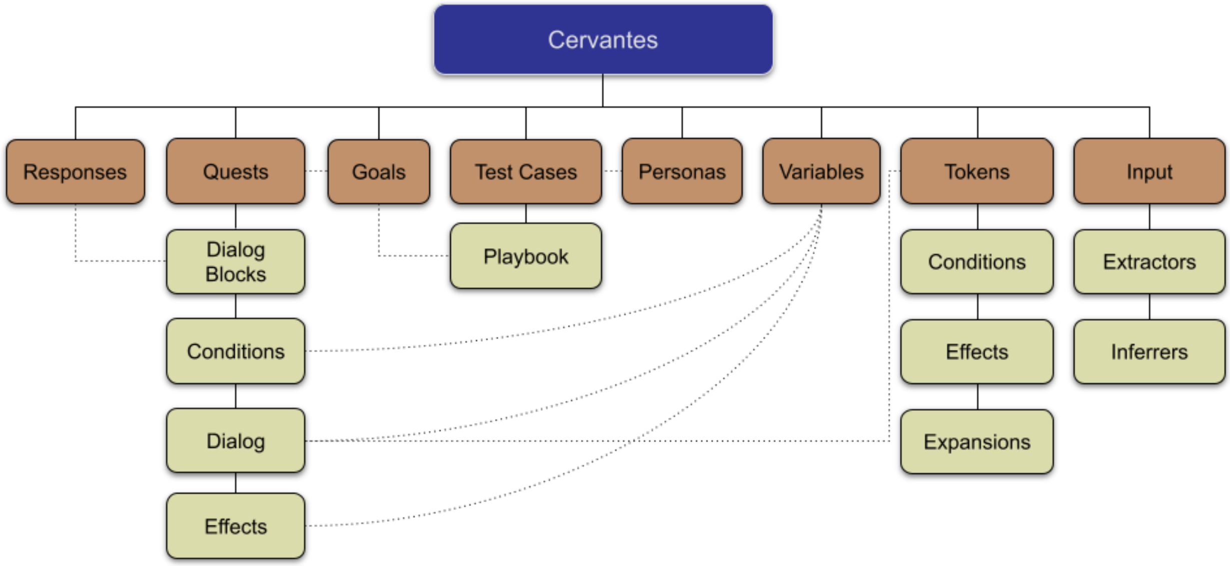


**Figure 4. Cervantes Functional Map Design.**

#### 3.2.5.1 Quest Authoring

Quest authoring is the most essential component of Cervantes. Cervantes' quest authoring functionality provides a subject matter expert with the tools to create dynamic and believable content quickly and easily for engaging with an interlocutor. Quest creation is broken down into the following components:

- **Quests:** Requests made by the SIENNA bot to perform an action
- **Goals:** The objectives to be obtained by each quest
- **Personas:** The pattern of life for the SIENNA bot, such as typical message response window (e.g., between 0900 and 1700), and response rate (e.g., within one hour)
- **Tokens:** Repeatable text blocks to add variability into the dialogue
- **Input Extractors:** Custom code-based elements for extracting essential message information

#### 3.2.5.2 Quest Testing and Evaluation

As quests are being created, having the ability to evaluate and test them is critical. To address this issue, we designed a simulator (an executing SIENNA-Bot) in Cervantes for quick evaluation of dialogue, extractors and flag captures without the actual sending of messages (they are all simulated).

Simulator interactions are designed around the concept of Test Cases. Each Test Case is an established interaction including all necessary conversation context and metadata such as:

- **Bot Initiated:** Defines the originator of the conversation, bot or human
- **Use Playbook:** Select a playbook for establishing a quest deployment strategy
- **Persona:** Select the desired persona for the bot
- **Message Modality:** The communication mechanism, SMS or Email
- **From:** Human-readable name for the conversation initiator
- **From Email or Phone Number:** Email address or phone number for conversation initiator
- **To:** Human-readable name for the receiver
- **To Email or Phone Number:** Email address or phone number for the message receiver
- **Subject:** The email subject line (not used for SMS)

Test Cases provide easy methods for:

- Crafting content, such as an email message and response
- Seeing the extracted flags as the conversation unfolds
- Understanding why quests / blocks were selected during the conversation

### 3.2.6 SIENNA-Bot

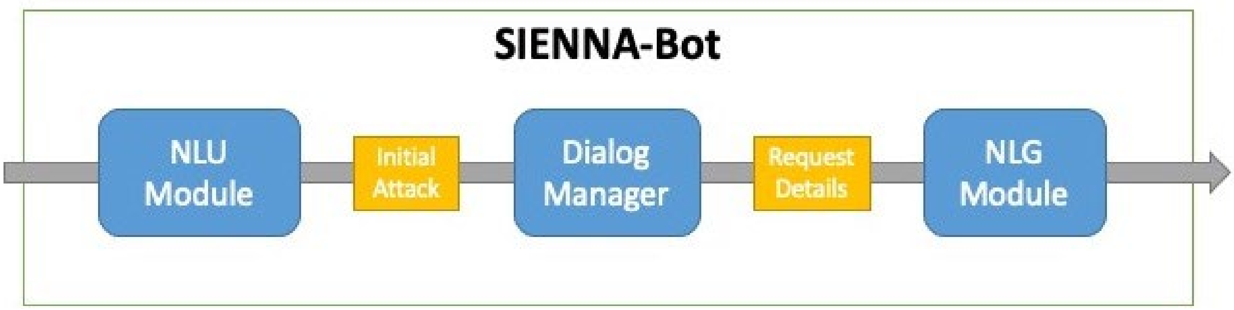


**Figure 5. SIENNA-Bot Architecture.**

The SIENNA chatbot is responsible for working with the generated content to produce coherent and convincing dialogue. This technology is composed of three main components that execute in a sequence:

- NLU Modules
- Dialogue Manager
- NLG Modules

The following sections describe the design of each of these components.

#### 3.2.6.1 NLU Modules

Critical for conducting conversational dialogue is the ability to parse and comprehend the messages received from the other party. To perform this task, SIENNA uses a series of NLU modules.

The objective of each NLU module is to translate an incoming message into:

- The conversation moves it performs
- The obligations it places on parties in the conversation
- The updates that it makes to the conversation state

SIENNA NLUs treat conversation moves as a form of speech acts [1]. By following spoken language understanding parlance, these acts can be conceived in terms of:

- Intents (moves and obligations)
- Slots (state components)

These products are then used by the Dialogue Manager to select the next move.

#### 3.2.6.2 Dialogue Manager

The goal of the dialogue manager is to reason about the best way to respond to a given input based upon the domain content, the information provided by the NLU module, and the current conversational state. The Dialogue Manager maintains an internal state of each ongoing conversation. The goal of the Dialogue Manager is to proceed through a sensible conversation based around the paradigm of a quest.

#### 3.2.6.3 NLG

Once the Dialogue Manager has determined the next move, it is up to the NLG to construct the response. This is done using a generative text system that leverages the domain authored content. The NLG system expands a context-free grammar that defines a set of possible text options and then uses the information provided by the Dialogue Manger to craft a valid response.

### 3.2.7 Trust Building

To effectively extract pertinent information from interlocutors, the SIENNA-Bot must be capable of effective elicitation. To achieve this objective, we leveraged the concept of trust. If the interlocutor trusts us, they are more likely to answer our questions and requests for personal information.

Trust is implicit in many online text conversations, and that trust can easily be betrayed through deception. Every conversation conducted by SIENNA is about deception. The interlocutor is trying to deceive the bot and gain critical information, while at the same time, the bot is trying to deceive the interlocutor to keep them engaged. To better understand how deception occurs during a conversation, our team conducted a study using the game Diplomacy where players negotiate through chats to forge and break alliances. This study relied on two technical components:

- A Game Engine (Backstabbr[1]) that lets users input their moves
- A Chat system that allows users to annotate if they are lying

This study led to some interesting insights into the concept of trust building, such as the fact that humans can detect lies at a rate better than random guessing, but only slightly better. It was also discovered that certain linguistic features appear to underpin successful deception. For example, words such as *true* and *honest* are likely to be perceived as truthful, yet they are also more likely to be deceptive. The team then used that information, and the data collected to develop a Long Short-Term Memory- (LSTM-)based architecture to process both the given message and the context (conversation history) in which the message was delivered. We expected this model to perform at a human level of lie detection, which just like humans, resulted in detection that was slightly better than random. Test set results are shown in Figure 6 and show that the neural model that integrates past messages and power dynamics approaches human performance.

| | ACTUAL LIE | | SUSPECTED LIE | |
|---|---|---|---|---|
| | **Lie $F_1$** | **Macro $F_1$** | **Lie $F_1$** | **Macro $F_1$** |
| **Baselines** | | | | |
| Random | 0.189 | 0.419 | 0.174 | 0.401 |
| Predict all as Lie | 0.214 | 0.107 | 0.186 | 0.093 |
| Predict all as Truth | 0.000 | 0.468 | 0.000 | 0.473 |
| Human | 0.243 | **0.574** | N/A | N/A |
| **Logistic Regression** | | | | |
| Language Features | 0.235 | 0.483 | 0.135 | 0.358 |
| Bag of Words | 0.184 | 0.525 | 0.142 | 0.516 |
| **Neural** | | | | |
| LSTM | 0.192 | 0.551 | 0.138 | 0.531 |
| LSTM + BERT | 0.216 | 0.559 | 0.203 | 0.542 |
| LSTM + Context | 0.263 | 0.555 | 0.174 | 0.532 |
| LSTM + BERT + Power | 0.224 | 0.564 | 0.230 | 0.530 |
| LSTM + Context + Power | **0.287** | 0.565 | 0.208 | **0.552** |
| LSTM + BERT + Context + Power | 0.257 | **0.571** | 0.156 | 0.534 |

**Figure 6. Test Set Results for Both our ACTUAL LIE and SUSPECTED LIE Tasks.**

In SIENNA, trust is a persistent metric for each conversant that is scaled between 0 (no trust, human is likely to immediately terminate the conversation) and 1 (human is cooperative and likely to continue the interaction). Various actions influence this trust estimate. For example, trust will drop for a human who repeatedly ignores an assigned quest. The Dialogue Manager uses trust as one of several signals when choosing how to deploy quests. For instance, if trust is high, the Dialogue Manager might continue to press for resolution of a currently assigned quest; whereas if trust is low, it might give up more easily.

[1] https://www.backstabbr.com/

By assigning quests a *difficulty level*, authors can signal how a quest should be used within the trust management system. For instance, if trust is low, the Dialogue Manager will try to assign *easy* quests to build up a rapport and increase trust. If trust has grown higher, the Dialogue Manager will assign *medium* and, subsequently, *hard* quests. This supports an authoring pattern for anti-scammer domains to convince a bad actor to agree to more and more difficult time-wasting tasks, as the "sunk cost fallacy" encourages them to believe they are getting closer and closer to closing a scam.

## 4.0 RESULTS AND DISCUSSION

The following sections discuss the approach taken to validate the methods above, along with a discussion of the observed results.

### 4.1 SIENNA Architecture

The SIENNA architecture can be conceptualized as author time and run time (See Figure 7). The following sections describe the key components that make up the architecture.

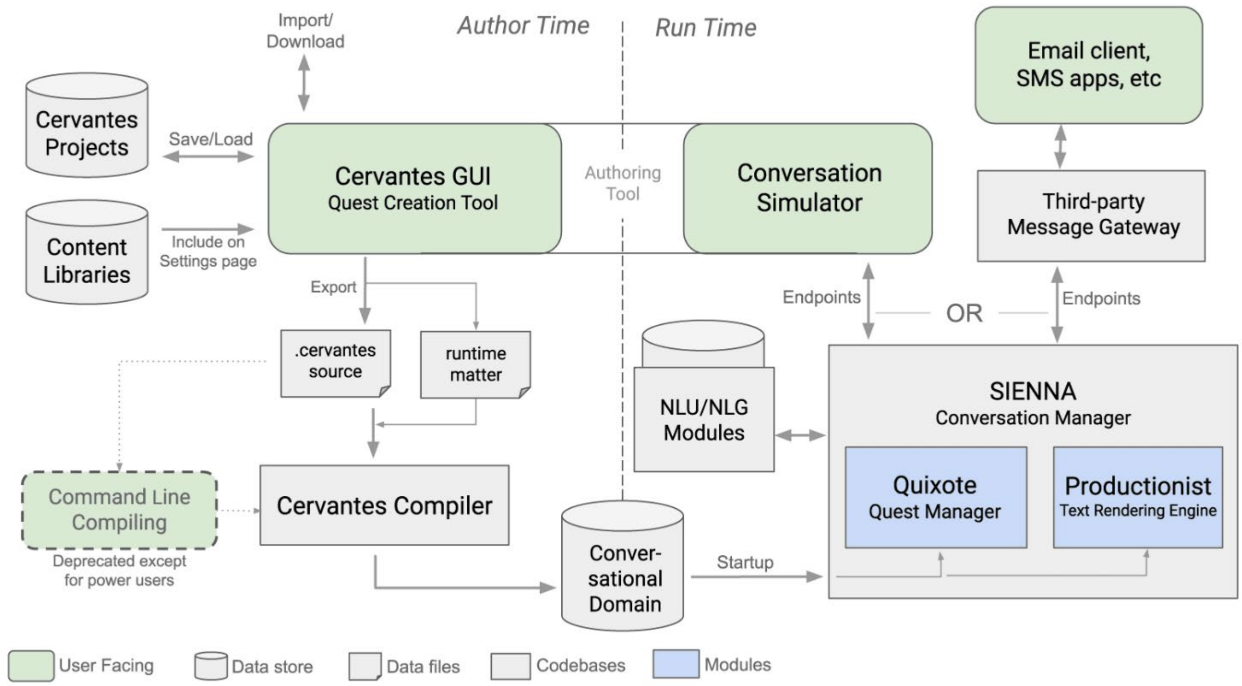


**Figure 7. Detailed SIENNA Architecture Diagram.**

#### 4.1.1 Author Time

Within the SIENNA architecture, author time is when content is created for consumption by the bot. Author time is composed of the following components:

- Cervantes GUI: The web-based front end for content creation
- Cervantes Compiler: The compiler for converting the human authored content into a format usable by the bot
- Cervantes Projects: A library of user authored content
- Cervantes Libraries: A library of user authored libraries. These include extractors and variables (described below)
- Conversational Domain: The compiled content consumable by the bot

In the initial version of the authoring tools, a command line compiling tool was used. This tool has since been deprecated and replaced with Cervantes, the GUI based authoring tool.

### 4.1.2 Run Time

The Run time components are the actual bots that interact with interlocutors, generating dialogue and extracting flags. The following components exist in Run Time:

- Conversation Simulator: A SIENNA bot designed to interface with Cervantes for quest testing and debugging
- SIENNA: The actual bot for interacting with a message handling gateway, such as those developed by the TA1 performers (e.g., SRI developed NEMESIS[2]) through which interactions with an attacker / interlocutor occur
- Quixote: The quest manager responsible for maintaining conversation state and constructing dialogue moves
- Productionist: The component responsible for generating natural language dialogue that will be delivered to the receiving party, building upon technologies from Expressionist (4.1.7.4)
- NLU/NLG Modules: Pluggable modules for extracting information from incoming messages and generating customized dialogue

The following sections discuss each of these components, starting with the overall design and approach for implementing quests.

### 4.1.3 Quests and Trust

As stated in hypothesis one (Section 2.0), by keeping an interlocutor engaged, we can occupy their time enough that they will not have time to phish other individuals.

To test this hypothesis, SIENNA developed the concept of Quests. As stated before, a *quest* is a task (or series of linked tasks) that the virtual target (SIENNA-Bot) asks the interlocutor to *complete*, as a way of wasting the interlocutor's time or capturing flags (i.e., collecting personal info) from the interlocutor. In order to have successful quests, we must first establish trust with the interlocutor. If they do not believe that we are willing to give them the information they initially are seeking, they will not participate in our quests as they increase in difficulty.

Any given quest has a *difficulty* based on how much time or effort we expect the interlocutor to take to complete. The ideal scenario is to maximize the number and difficulty of the quests the interlocutor completes during an interaction. Quests that force the interlocutor to take real-world actions away from their computer are the most difficult and thus desirable.

To determine how to handle human responses to quests, we first identified a number of possible ways an interlocutor assigned a quest might respond:

- Completing it
    - Example: "No problem, here's that form you asked me to fill out."

[2] Natural Language Engagement of Malicious Entities through a Social Interaction Service

- Pretending to complete it, or saying they'll do it but never following through
  - Example: "I've finished the form and I'll send it later."
  - Example: "I can get that form filled out later tonight, but first..."
- Partially completing it
  - Example: "I filled out the first page of the form, but haven't done the rest."
- Questioning it
  - Example: "Why do you need me to fill out this form?"
- Bargaining about terms
  - Example: "I'll fill out the form, but first you need to send me your bank account info so I know you're serious about this."
- Refusing to do it
  - Example: "I'm not filling out this form."
- Ignoring it
  - Example: "Let's get back to this money transfer I mentioned..."
- Terminating contact
  - Example: (no response)

In general, we can place these responses in three categories: *Executing* the quest, even if only partially (most ideal); *Haggling* about the quest's details; and *Refusing* the quest (least ideal). We want to avoid haggling interactions since this involves more complex natural language understanding and generation requirements: SIENNA would need to parse, comprehend, and reuse specific details, and risk revealing its identity as a bot. Because of this, we chose a strategy where we prefer to unassign quests (with a message such as "you know what, never mind about that") if interlocutors do not seem immediately compliant.

The best metric for predicting an interlocutor's response is their **trust** for the virtual target. An interlocutor who trusts that the target is a real human (and one likely to fall for their scam) will be willing to complete more quests, and complete more difficult quests. If trust falls to zero, however, the interlocutor gives up on the target, possibly because they realize the target is a bot.

We can detect signs of falling trust when interlocutors haggle over or refuse quests; when messages become much terser; or when the NLU component detects signs of anger, frustration, or impatience in the interlocutor's messages. For instance, from a corpus of human scam-baiters, we could often identify patterns corresponding to indicators of high and low trust by simply looking at words with positive and negative valence as shown in Table 1 (for more details on our final implementation of this, see Section 4.1.8). Bolded words have positive or negative valence.

**Table 1. Examples of Attacker Sentences Illustrating Different Levels of Trust.**

| ***Attacker Sentences*** | ***Trust Level*** |
|---|---|
| I am **glad** to receive your response and hereby forward to you an **agreement**/bond. | High Trust |
| **Thanks** for your **good** response to me, anyway I must tell you that **I am really comfortable** with you over this my transaction | High Trust |
| I got your email **but** I was quite **disappointed** at the way you talk. | Low Trust |
| I **cannot** send my picture to you until I am convinced that we are in business. | Low Trust |

Assigning a quest, or asking for a flag, is a risky act in that it may reduce trust by making the interlocutor suspicious, or by making it seem less likely to them that their scam target is viable. This danger increases with the difficulty of the quest or the sensitivity of the flag requested. However, a quest generator that monitors and reasons over trust can be more considered in how it assigns quests or asks for information, resulting in a more prolonged engagement with the interlocutor.

Based on the above, we designed and built our quest management ecosystem to achieve the following goals:

- Interface with SIENNA and other existing components of ASED
- Allow for authoring a library of human-created quests tagged with difficulty levels
- Begin a conversation by assigning simple quests to "hook" the interlocutor and build trust
- Introduce more difficult quests as trust increases, eventually working up to difficult real-world quests as the interlocutor's "sunk cost fallacy" makes them more likely to complete them
- Maintain a basic dialogue about assigned quests:
    - Acknowledging an interlocutor's message that they have started, completed, or refused a quest
    - Prod an interlocutor about an unfinished quest
    - Recognize when the conversation manager cannot understand well enough to reply sensibly and deflect onto a new topic
- Attempt to increase trust if signals are received that trust is declining by:
    - speaking in a more compliant/accommodating manner
    - unassigning a quest
    - assigning easier quests

### 4.1.4 Quest Validation

There are several possible ways an interlocutor might "complete" a quest:

- **Correctly** completing the quest as assigned
- **Partially** completing a quest (e.g., sending a phone number without an area code)
- **Cheating** by sending something that appears to be what is requested but isn't
- **Failing** by responding with something that clearly isn't what was requested

In theory, a virtual target should be able to distinguish between these states, but we believe that in practice this is less important that it seems. An interlocutor who "cheats" or sends a partial response has still wasted some amount of time, perhaps on a forgery or plausible-seeming false response. Additionally, from an interlocutor's perspective there is little motivation to "fail" at an accepted quest, since a real human (which they perceive the target to be) would detect the failure and reject the response.

However, some quests with textual responses can still be validated, and this is worth doing when possible, for verisimilitude (to help maintain trust that the target is a real human). For instance, if we ask for a phone number, we might check that it has the right number of digits and is in a plausible area code based on any geographic information the interlocutor has previously divulged. We can also "fake" verification questions on other kinds of data we aren't actually analyzing: for example, if we requested the interlocutor send a photo of themselves, we can plausibly ask if the photo is really them, without doing any kind of real analysis on the attached file.

#### 4.1.5 Quest Taxonomy

In addition to general stalling or delaying responses, we identify several categories of quests.

##### 4.1.5.1 Trivial Quests

Quests that can be completed with a simple reply; this is basically just a delaying tactic. There are many possibilities here, but a few include:

- Asking for clarification or more information
- Target pretends they've already been contacted by someone similar, asks if the interlocutor is from the same company/organization
- Target pretends the interlocutor missed a message from them, asks them to check their spam folder, etc.
- Target asks for a reminder message to follow-up at a later date/time

##### 4.1.5.2 Flag Capturing

Similar to trivial quests but with the goal of capturing flags about the interlocutor.

- Asking the interlocutor to identify their time zone so they know when is good to email
- Asking the interlocutor to identify their gender so they know what form of address to use
- Encouraging the interlocutor to click a link so Natural Language Engagement of Malicious Entities through a Social Interaction Service (NEMESIS) can capture additional information

#### 4.1.5.3 Administrative Hoops

Asking the interlocutor to do annoying time-wasting digital activities to verify their sincerity or identity.

- Target says they need the interlocutor's company's Data Universal Number System (DUNS) number, provides instructions for signing up for one at Dun & Bradstreet if they don't have it. (Can check the provided number has the right number of digits etc. as a validation step.)
- Pretending to originally be from the same town in the interlocutor's address, and asking them vetting questions like "What's your favorite restaurant around there?"

#### 4.1.5.4 Switching Modalities

Delays based on switching to a different communication method (which also captures a flag re: phone number, account name, etc.).

- Requesting texting to verify identity, scheduling this for a certain time, then a back and forth before returning to prior communication channel
- Asking the interlocutor to install a particular communication app/technology (Kik, Signal, Pretty Good Privacy [PGP] etc.) and then pretending to have trouble connecting there, eventually giving up and going back to regular email

#### 4.1.5.5 Document Creation / Forgeries

Getting the target to waste time making fake documents.

- Requesting a formal document outlining the bill/service etc. under discussion, then after it's delivered asking it to be redone with particular requirements "from our office manager" (font size, margins, page dimensions, document format, etc.)
- If not starting on LinkedIn, asking to see the target's LinkedIn page (they might need to make a fake one)
- Expressing passion for a particular charity and asking the interlocutor to make a small donation as a token of their goodwill, and provide proof (interlocutor either does this or spends time faking a receipt)

#### 4.1.5.6 Real-World Time Wasters

Getting the interlocutor to step away from their computer to do something more involved. These are the most "difficult" and thus desirable quests to assign, but also tricky to plausibly justify and require a high amount of trust.

- Asking for a photo of the interlocutor with a sign showing their name. (This is a frequent ploy of scam baiters, so may be a red flag to scammers to stop engaging.)
- Claim the scammer's internet protocol (IP) is being blocked: can they try responding from a different physical location?

- Asking for a quick phone call to verify their identity, scheduling this for a certain time, then send a series of messages asking why they aren't picking up, etc., then suggesting the interlocutor move to a new physical location with better reception (e.g., "try to get by a different cell tower")
- Asking for a phone call as above but stipulating it must be a landline.
- If the scam involves a pending financial transaction, claiming the target's bank can't transfer money to the interlocutor's bank: can they set up a new account at a different bank to receive the funds?

#### 4.1.5.7 Domain-Specific Quests

While many quests are generic and applicable across a variety of scams, those customized to a particular scam are good for increasing believability and trust. For the domain of a "work from home" job offer scam, for instance, some domain-specific quests might be:

- Target requests a formal job offer on company letterhead, can quibble with format/details
- Target asks to see an official website for the hiring firm listing the interlocutor's name as an employee, which might inspire the interlocutor to create a fake website with this info
- If a photo of a check to be cashed is sent (common in this scam), target claims their bank needs it in a different/rare format like: Thermogravimetric Analysis (TGA), higher resolution (300 dpi), separate images for check front/back, etc.
- Target needs hard-to-find info about the job for their records, like its Internal Revenue Service (IRS) Position Classification Code

For the Q1 2021 evaluation at Virginia Tech (VT), we created a library of roughly 60 quests instantiating many of these ideas. Ten of these were specific to the VT domain, with the others more reusable. Later, we turned them into a "Default Quests" library in the CervantesDSL.

### 4.1.6 Author Time: Cervantes GUI

In Section 3.2.4, we covered the details of the DSL developed for capturing and encoding the human-authored content. Once the DSL was finalized, we used the semantics as the foundation for building a user-friendly quest authoring tool, *Cervantes*.

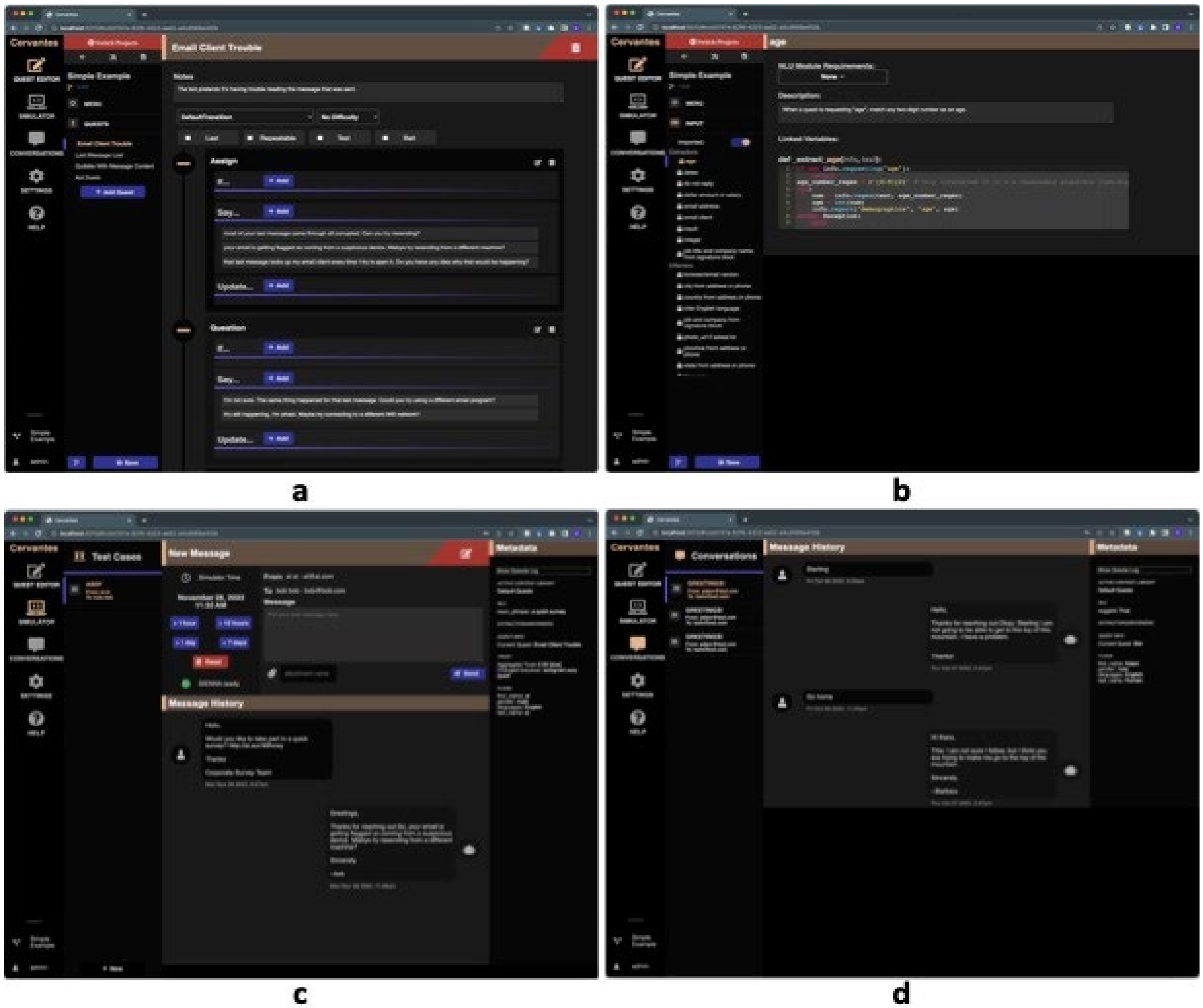

**Figure 8. Cervantes UI.**

Our design goal for Cervantes was to create a user-friendly editor and test platform. We achieved this by creating a modern browser-based user interface that uses design strategies and familiar design patterns that users can pick up quickly. The core of the tool is written in JQuery and raw JavaScript along with some imported plugins to support various UI features.

The UI features a core set of functions which are present on the leftmost navigation bar of the interface (See Figure 8) these are:

- **Quest Editor:** (Figure 8a) Allows a user to create valid CervantesDSL content without actually understanding the underlying syntax. This content can then be used in the Simulator to test how the SIENNA-Bot will interact with actual interlocutors
- **Simulator:** (Figure 8c) Allows for the creation of Test Cases which use the content created in the Quest Editor and allows the user to simulate any number of moves or different scenarios; including bot-initiated attacks

- **Conversations:** (Figure 8d) Allows the user to visit previous conversations that took place by SIENNA-Bot running through its API or conversations that took place in the simulator
- **Settings:** Allows users and administration to change their current settings like password, team membership, email, username, and allows admins to add new users to the system, export all content, and create new teams
- **Help:** Forwards the user to our documentation which can be used as reference to learn how to use the tool

The Quest Editor utilizes the full set of CervantesDSL features, including Quests, Goals, Tokens, Variables, Responses, Personas, Inferrers, and Extractors. Each of these features can be found in the project navigation panel, which appears after opening a project in the Quest Editor. All features have a custom UI which is tailored for editing and creating that type of content.

The editor allows the user to define any number of Quests that the interlocutor will be presented with given certain conditions and state maintained by the underlying dialogue manager (Quixote). Each quest can contain a name, notes, difficulty, goals, triggers and any number of dialogue blocks (Responses). Each dialogue block can contain *conditions* in which the quest will be triggered, *effects* that take place after it is completed, and any number of *say blocks*. Conditions can trigger based on variables or the input text from the interlocutor. Effects can update the conversational state, set variables, or end the current quest and move on. Say blocks can contain tokens, variables and bot invocations. For more details on quest authoring, see the Cervantes User Manual.

To improve usability, we use a JQuery plugin Tagify which allows users to have auto complete for these items after typing @ for Tokens and # for variables.

Cervantes also includes an editor for creating a set of goals. Goals allow the user to define what concrete objectives they have for the project. Each goal is a free form string which can then be tagged onto a quest. When a quest is completed, the goal will be marked as complete. The user can create any number of goals.

Tokens allow the user to view imported tokens from our extensive examples or define expansions which can be used in say blocks throughout Cervantes. Each token has a name and any number of expansions. Each expansion can contain text, other tokens, variables, or bot invocations. Each expansion can have conditions (when this expansion can appear), effects (what happens after this expansion appears) and a weight which will allow for that expansion to appear more often. When a token is used in a say block you can click on the tag and see examples of what that token may appear as, this can also be achieved by clicking the dice icon on the token editor screen.

Variables allow the user to view imported variables from our base libraries or to define custom state variables that can appear in say blocks, tokens, responses, and can be set in Inferrers or Extractors. Variables contain a name, description, type and initial value if the variable is not runtime only.

Responses allow the user to view the built-in dialogue block types or to create new types of dialogue blocks. Responses with an open lock icon can be edited, but not removed, as they are required by the CervantesDSL. Responses contain a name, transition (token), conditions, and any number of say blocks.

Personas allow the user to create different patterns of life that the SIENNA-Bot will use to respond to the interlocutor in various ways. Personas can include a name, availability times, time zone, responding delays, and follow-up settings. After a persona is created it can be used in a test case to simulate these new patterns of life.

Input allows the user to view the imported Inferrers and Extractors from our base examples as well as create new ones. Since Inferrers and Extractors use Python code to run, this view features a full-service inline code editor and highlighter. For this we used *Ace,* an open-source JavaScript plugin which is lightweight yet fully featured. An Extractor or Inferrer can contain a name, status, NLG module requirements, description, and python code. While writing code, the user can click one of our example interaction buttons which add example code to do the most common interactions with our API including: setting a variable, checking message content with regex, and getting the value of a state variable.

The Simulator uses the currently loaded content created in the Quest Editor and test cases to simulate conversations. A test case contains a from name, from email, to name, to email, subject, Persona, Playbook and bot-initiated settings. For each project, the user can have any number of test cases, allowing the user to simulate various patterns of life or different types of attacks. If the user has defined goals for your project, they can create a playbook for this test case using an ordered or unordered list of existing goals.

When simulating a test-case, the user can virtually pass time in set increments using the time settings on the left side of the message screen. The user can view the current time of your conversation directly above these buttons or reset the current conversation below. When interacting on the message screen, the user can send text messages of any length. If they wish to simulate the inclusion of an attachment, they must enter the attachment name including the extension before sending the message. After the SIENNA-Bot has responded, the user can view complete state information in the metadata column including the current quest being used, flags, trust levels NLU, and the Quixote log.

#### 4.1.6.1 Dialogue Construction

Essential to Cervantes is the ability to construct quests. Understanding how the SIENNA-Bot utilizes the authored content is essential for creating convincing dialogue. When dialogue is constructed, a specific pattern is followed. Figure 9 illustrates a flowchart of dialogue construction.

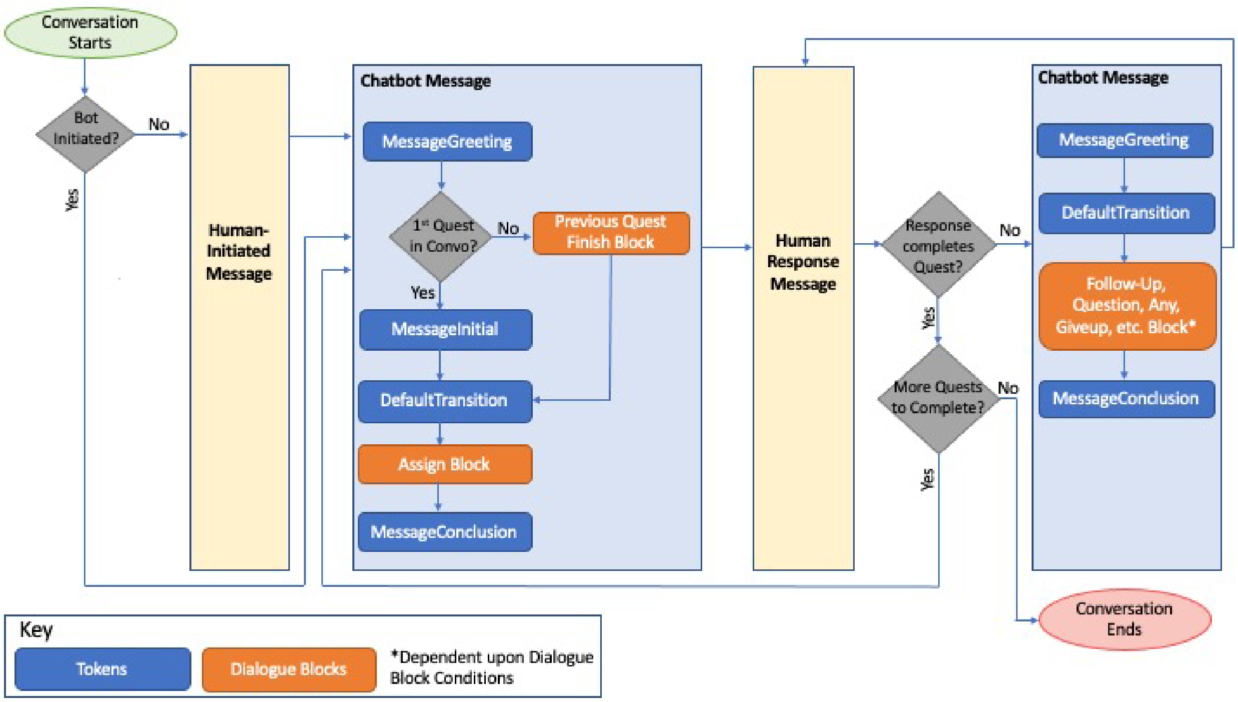


**Figure 9. Dialogue Conversational Flowchart.**

A SIENNA bot can either start (offensive) or respond (defensive) to a conversation. The only impact this has on dialogue generation is if a canned initial start message is used, or if the content of the received message is used to craft a response.

Cervantes has a defined set of tokens that are used to facilitate conversation construction. These tokens: *MessageGreeting*, *MessageInitial*, *DefaultTransition*, and *MessageConclusion* are designed to simplify the creation of dynamic content. The flow in Figure 9 shows how these tokens are combined to construct dialogue. The Cervantes User Manual has a detailed description on how to utilize these tokens for optimal quest creation.

#### 4.1.6.2 Rights and Roles

Once team performers started using Cervantes it became clear we needed to incorporate better rights and roles for controlling project access. To achieve this goal, we modeled our rights and roles based on the same process used by GitLab. In this model, there are three roles: User, Developer, and Admin. Each role has distinct rights within the interface (See Figure 10).

**General Permissions**

| | User | Developer | Admin |
|---|---|---|---|
| Use Advanced Features | X | X | X |
| Create Teams | X | X | X |
| Create File Versions | * | * | X |
| Delete File Version | * | * | X** |
| Save File Changes | * | * | X |
| Link Team and Project | * | * | X |
| Unlink Team and Project | * | * | X |
| Add User to Team | * | * | X |
| Remove User from Team | * | * | X |
| Delete Team | * | * | X |
| View all Projects | | | X |
| Edit all Projects | | | X |
| View all Teams | | | X |
| See all Team Users | | | X |
| Create Personal Project | X | X | X |
| Delete Personal Project | X | X | X |
| Create Team Project | * | * | X |
| Delete Team Project | * | * | X |

**Team Projects**

| | User | Developer | Admin |
|---|---|---|---|
| Save File Changes | | X | X |
| Create Team Project | X | X | X |
| Delete Team Project | | | X |
| Link Team and Project | X | X | X |
| Unlink Team and Project | | | X |
| Add User to Team | | | X |
| Remove User from Team | | | X |
| View Team Users | X | X | X |
| View Team Projects | X | X | X |
| Create File Versions | | X | X |
| Delete File Version | | X** | X** |

* See Team and Personal Project Roles
** For file versions you own in projects you can view

**Personal Projects**

| | User | Developer | Admin |
|---|---|---|---|
| Create Project | X | X | X |
| Delete Project | X | X | X |
| Create File Versions | X | X | X |
| Delete File Versions | X | X | X |
| Save File Changes | X | X | X |

**Figure 10. Cervantes Project Right and Roles.**

Using this model, a single Cervantes instance can host many teams without concern of exposing sensitive projects, or having content changed or deleted by non-team members. For details on using the rights and roles, see the Cervantes User Manual, or the integrated Cervantes help.

#### 4.1.7 SIENNA Bot

SIENNA is a hybrid dialogue system that combines classical symbolic AI techniques with cutting-edge neural architectures. The SIENNA-Bot is a Python based application built on the Flask framework. All communications with SIENNA occur via Hyptertext Transfer Protocol (HTTP) REST calls. SIENNA currently contains two different API interaction sets.

- ***Complex API*** is geared towards supporting the larger ASED campaign infrastructure and interfacing with NEMESIS
- ***Simple API*** a basic API currently used between the Cervantes Simulator and the SIENNA bot

The following sections describe the design and details of the SIENNA bot.

##### 4.1.7.1 Complex API: Interacting with NEMESIS

Our software was just one component of the larger ASED program. As such, it was critical that we implemented a set of APIs to interact with the other components. The *complex* API, so called because it provides much more detail and touch points then the *simple* API described below, is illustrated in Figure 11. In this API, there are three primary touch points into SIENNA:

- Reply2: The endpoint responsible for starting conversations and relying to messages
- Wakeup: Used by the larger system to wake-up the bot and have to check for messages

- Sent: Used by the larger system to inform SIENNA that a message has been delivered

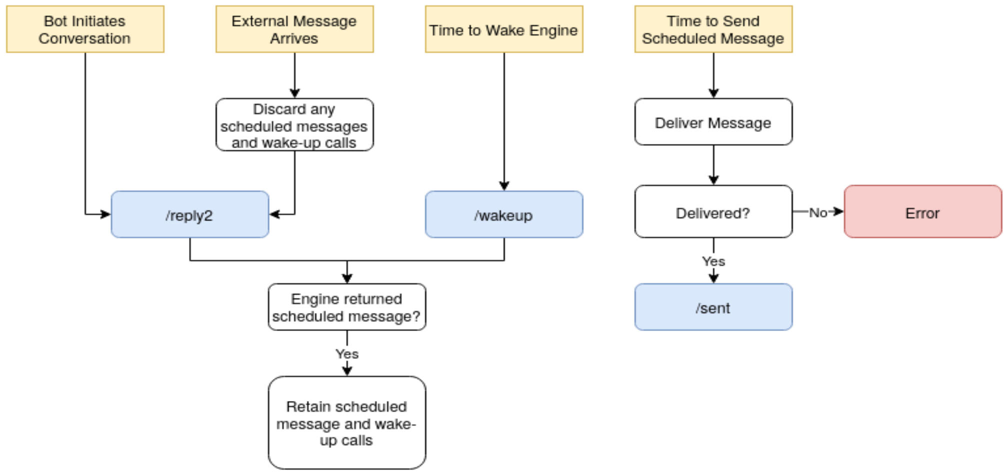


**Figure 11. Complex API Pipeline for Interacting with SIENNA.**

#### 4.1.7.2 Simple API: Interacting with the Simulator

To evaluate content as it is created using the authoring tool, we incorporated a simulator. This simulator is actually just a single SIENNA bot. However, to perform the desired simulation of conversations, the full complexity of the complex API was no required, so we developed the simple API (See Figure 12). This interface uses a much smaller data structure than the complex API and has slightly different endpoints.

- Begin: Starts a new conversation with the bot
- Continue: resumes a conversations
- Delivered: Informs the bot that a message has been delivered

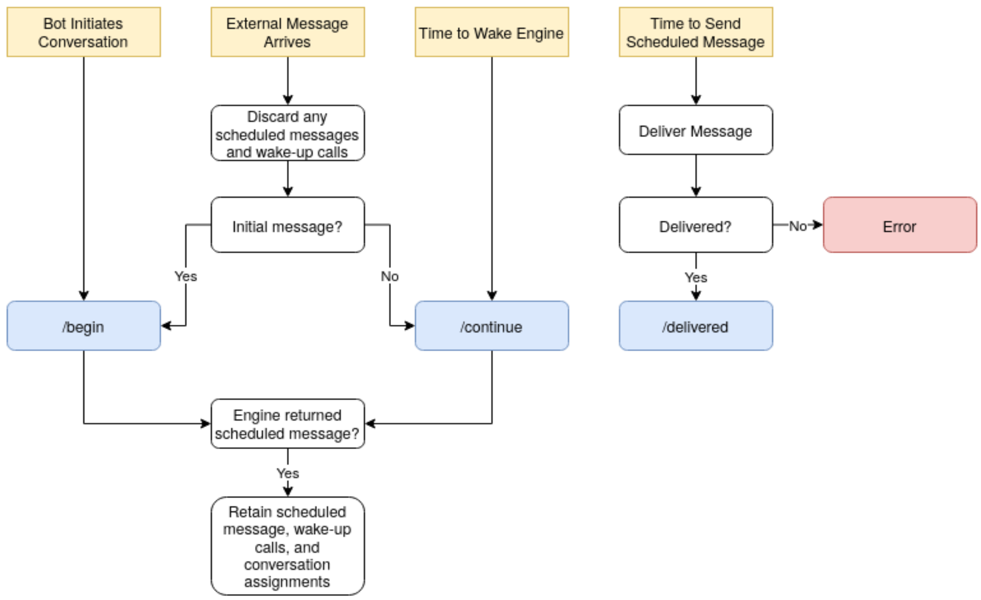


**Figure 12. SIENNA Simple API for Interacting with the Simulator.**

#### 4.1.7.3 System Engineering (SE)

Essential to the successful operation of SIENNA is a well architected and implemented system. This section describes the system design considerations and highlights the issues encountered and how they were overcome during the program.

##### 4.1.7.3.1 Global Bot Blackboard

The first major systems functionality introduced into SIENNA was the ability to support multiple simultaneous email threads. Because the program-wide objective required the ability to defend multiple targets against multiple interlocutors, the SIENNA solution must be capable of supporting the modeling of multiple simultaneous email threads. Because the aim of SIENNA was to model coordinative behaviors in which bots reason over the content of other threads and carry out coordinated conversational activity across threads, this design required careful consideration. SIENNA must explicitly model all ongoing threads in terms of the information that is required for reasoning. The resulting architecture supported the functionality for modeling multiple simultaneous threads and maintaining information about each thread in a single global blackboard state that is accessible to all bots.

However, during the Winter 2020 evaluation, the scale of attacks was far greater than in any of the previous evaluations. This equated to many more conversations and some conversations comprising hundreds or thousands of messages. In one instance, frequent messages from an automated service were falsely identified as an attack which led to SIENNA having to process a conversation that grew to hundreds of messages. Because the design of state within SIENNA consisted of reconstructing the global state for each system call, the scaling necessary for the quantity of messages quickly broke down. To address this issue, we overhauled the SIENNA state management functionality such that each conversation maintained its own state. This resulted in much smaller state data objects.

#### 4.1.7.3.2 Scheduled Message Delivery

In the initial proof of concept, email correspondence was modeled as a real-time conversation. When one party sends a message, the corresponding party immediately responds. This pattern of life is not indicative of real-world email communications where responses can occur over the course of hours and days. To address this issue, SIENNA employed a technique where bots exhibit human behaviors such as not sending emails in the middle of the night or on weekends. The SIENNA author can specify bot *personas*: content generation rules for a given email message such as when the delivery should occur (n hours from x).

#### 4.1.7.3.3 Inactivity System Wakes

The initial SIENNA prototype only responded to received messages. However, this approach does not properly correspond to actual email behaviors. It is not atypical for one party to send multiple emails in a row before receiving a response. Because of the "call and response" nature of our original design, the SIENNA bots did not have the ability to generate and deliver multiple messages. To address this shortcoming, we incorporated *wake-up calls*. With this design, each generated message comes attached with a time at which SIENNA should be woken up. When the bot is "woken up" for a given thread, it will then reason over the current state and possibly generate a new message. This design approach allows the bot to pull a silent interlocutor back into the conversation after a period of inactivity.

#### 4.1.7.3.4 End State Criteria

In the initial design, SIENNA had no concept of ending a conversation. This oversight resulted in SIENNA continuing to generate and send messages when the interlocutor was clearly finished. To address this oversight, we incorporated end-state conditions into the bot. These included rules such as end the conversation after four unanswered questions. We further extended end state detection to identify words such as "unsubscribe" and "stop" as markers that the interlocutor would like us to stop communicating with them. The final addition to end state criteria was the incorporation of an affordance within Cervantes that would mark a dialogue block as the final exchange for the conversation. This addition addresses the case where the bot is finished conversing (i.e., it has acquired all of the necessary information) yet the interlocutor still sends messages. In this case, the bot will just ignore all subsequent messages.

#### 4.1.7.3.5 Offensive Attacks

The initial design of the SIENNA bots was to react and engage with interlocutors. Over the course of the program, interest grew to also have the bot take an offensive stance where the bot would initiate a conversation. Due to our design of SIENNA, incorporating the functionality to allow the bot to either respond or initiate was a straightforward task. The biggest technical hurdle was the setting and sending of the initial message. The design of the system is such that the incoming content is used to generate responses. Without having an initial message to react to, we created and incorporated functionality to mark and select initial messages for dissemination. Once we made this change to the construction and initiation of conversations, the rest of the conversational flow was essentially the same as a defensive bot.

#### 4.1.7.4 Dialogue Generation

To generate content, the SIENNA team adopted the techniques used for videogame procedural content generation refined by Dr. James Ryan and his *Expressionist* tool [2]. This approach uses a novel authoring scheme for natural language generation that is driven by attribute grammars. Attribute grammars [3] are a computational formalism that modifies context-free grammars by introducing tags that are attached to elements within the grammar. These tags are then attached to the nonterminal symbols in a grammar during text generation. Using this approach, we created an initial proof of concept bot, the *Bored and Lonely* bot. The goal of this bot was to counter impersonation attacks by having the bot evolve into either a needy narcissist or a spiteful colleague persona.

The needy narcissist persona loves to gossip about coworkers and colleagues in the targeted field. This bot expresses distastes for recent colleague publications and likes to tell rambling stories about themselves. Because narcissistic personalities tend to be bad listeners, the bot itself doesn't require much knowledge about the content of the dialogue to effectively present as a human. The spiteful colleague persona bot leverages a narrative conceit to introduce a fictional backstory as a means of conversation. Because the interlocutor is only pretending to know the subject, using a fictional story allows the bot to quickly take control of the conversation without having to understand much of the content.

One of the key benefits of this bot style is that it allowed the SIENNA team to quickly construct a proof-of-concept bot without having to develop the necessary components for parsing and comprehending text content. This allowed a more focused approach to the dialogue generation capabilities and its shortcomings. One of those identified shortcomings was the inability of the dialogue generator (DG) to maintain any form of state. State within a dialogue generator is the retaining of specific words or phraseology. For example, if a person always starts off an email message with the phrase “Hello there,” then it is important that the bot maintains that style to promote realism. To address this issue, we enhanced SIENNA to incorporate state into the DG. As a result, once a specific “voice” is determined for a bot, that style of interacting is maintained throughout the conversation. State is maintained in the DG by using *preconditions* and *effects*. Preconditions are used to determine if a production rule should or should not be used as part of the current dialogue text. And once a rule is successful, any added effects are used to update the state. This addition improved upon Expressionist by including the fine-grained authorial control of production systems [4] while retaining the semantics-to-text power of attribute systems.

The next big revision to the DG system was the refinement of the computational formalism used for determining conversational strategy. For this change, we implemented the logic to represent a bot's total set of conversational strategies as a finite-state machine (FSM) in which each state represents a distinct conversational strategy (e.g., pretending to struggle with an attachment). Using this approach, a bot can then transition between various states if certain aspects of the conversational state holds. For example, if the interlocutor includes a link in their message, the bot can transition to the *struggle with the link* state. Each state in the machine (conversational strategy) is then structured as a partial-order plan [5] where the plan steps correspond to individual conversation moves that the bot can perform. Before a turn, all available plan steps are scored using utility-based action selection. This scoring distribution is then used to determine which step will be performed. If the selected step is within a different FSM state, a state transition occurs.

The next addition to the DG was the incorporation of custom dialogue generation engines. The objective here was to explore the ability of using NLU modules that could negotiate or bargain. The resulting product was a NLG called *Specialist*. Specialist can generate dialogue based on the semantics of the input message by *quibbling* (argue or raise objections to a trivial matter) with the interlocutor. This model is a Generative Pre-Trained (GPT)-based model. To train the model, dozens of conversations with requests and quibble-like responses were authored in Expressionist. These conversations were then used as input into GPT-3 to augment the conversation corpus, producing over ten-thousand quibble-like responses. We then incorporated the Diplomacy Corpus to augment the conversational data focused on quibbling. Because the GPT-3 model was too large to realistically deploy within SIENNA, we retrained with GPT-2 to generate a model capable of generating quibble-specific responses. With this addition, SIENNA content authors can now leverage Specialist to provide quibbling as an effective conversational strategy. For example, consider a situation where an interlocutor quibbles over the details of an assigned quest. Here, the content author can pass the interlocutor's quibbled content into Specialist and get back sufficient dialogue to respond. We then further expanded Specialist to provide dialogue for *deflecting* and *answering questions*.

We developed another NLU within the DG is for the task of classifying discourse-acts, *Comprehensionist*. The key benefits of a discourse-act classifier are: a) provides dialogue state information, and b) provides affordances for the quest author to author content specific to certain discourse-acts. For example, by detecting that an interlocutor is arguing when responding to a quest, SIENNA could use that information to either change the direction of the quest, or assign a new, possibly easier, quest.

Once Comprehensionist was incorporated and proved effective, we explored other areas for discourse-act classifiers. This resulted in *Completionist*, a discourse-act classifier capable of understanding whether the interlocutor is trying to elicit information or elicit an assessment that falls into one of four categories:

- Yes/No
- Too much information
- Opinion
- Factual

#### 4.1.7.5 NLU Modules

Another critical component for conducting conversational dialogue is the ability to parse and comprehend the messages received from the other party. To perform this task, SIENNA relied upon a series of NLU modules. The extent and capabilities of these modules grew throughout the duration of the program. This section documents that progression.

To provide comprehension of an interlocutor's messages, we chose an NLU approach that could leverage the capabilities of Expressionist and the various aspects of the DG. Specifically, the design included a grammar whose outputs are messages that we imagine an interlocutor would send. By reusing a subset of the tags that are part of the bot grammars, the output of the attack grammar aligned with the tags that also drive the operation of the DG. This in turn allows us to generate annotated training data for a *sequence-to-sequence* [6] task using LSTM technology.

Using this approach starts first with translation. A natural language interlocutor message is translated into a sequence of tags from our grammar. This translation captures the essence of the message with regard to the core concerns of the dialogue manager. Named entity recognition and LSTM copy mechanism [7] are used to form special structured tags that map substrings in the received message into specific recognizable concerns. For example, if an interlocutor references a colleague, the colleague's name could be copied from the input sequence and pasted into a corresponding slot in a special tag dedicated to colleague mentions.

To explore the potential of this design approach, we developed a pipeline that converts message tokens into word embeddings using the Embeddings from Language Models (ELMo) model [8] and then concatenated those representations with *part-of-speech tags* and *named entities* produced by applying other off-the-shelf tools to the token sequences. To test and validate our approach, we used our Expressionist tool to generate training data within the *gift card scam* domain and then had the system conduct conversations with itself. Using this approach, we were able to collect hundreds of thousands of procedural conversations. The transcripts from these conversations were then automatically annotated for training. The resulting model from this training produced a workable demo that was highlighted at the ASED August 2019 Workshop in Dayton, Ohio (See Figure 13).

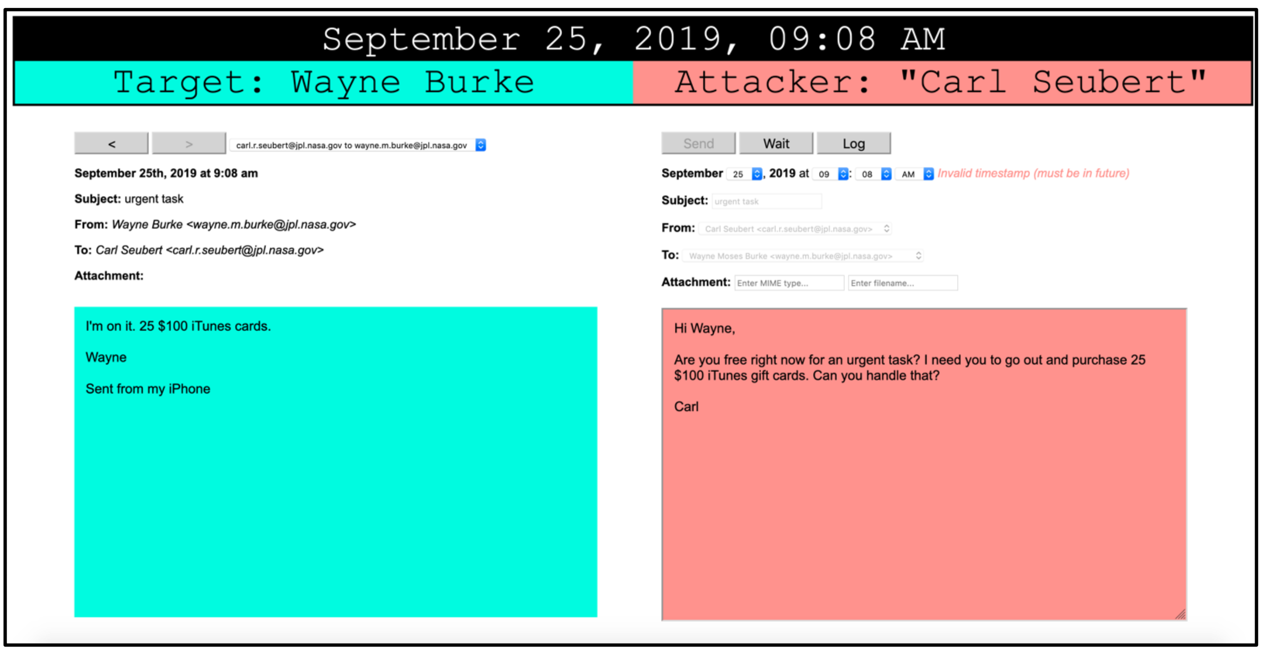


**Figure 13. Gift Card Scam Demo Screenshot Highlighting NLU Technology at ASED August 2019 Workshop.**

The objective of the NLU within SIENNA is to a) understand the pragmatics of what the interlocutor is saying, and b) extract critical pieces of information from the interlocutor's messages. Working from the knowledge obtained during the workshop, and aligning with these objectives, we turned our focus from an LSTM-based architecture towards a pre-trained Transformer model (Bidirectional Encoder Representations from Transformers (BERT), Robustly Optimized BERT Pretraining Approach (RoBERTa) [9]). We then added extra layers to the model for handling the primary objectives of our technology (understanding the pragmatics of the interlocutor's message and extracting critical pieces of information from the message). With this change in architecture, we were able to leverage the semantic knowledge captured in the pre-trained model with a vastly reduced training time by fine-tuning, instead of retraining from scratch. This technique allowed us to *patch* models with newly acquired information. To verify the effectiveness of our patching technique, we generated new content within the gift card scam domain. For this, we used Expressionist to generate many examples of an interlocutor changing the denomination, and then fine-tuned the model by training just on this newly acquired data. This approach added robustness to the model and resulted in a compelling demonstration of a novel technique for NLU model training.

We also made a change in content processing. Previously we were only processing the interlocutor message. Now we also process, in tandem, a representation of the recent history of the conversation. This technique proved especially useful in cases where the content of a message was ambiguous when divorced from the current conversation context, such as a message response of just "Ok, Thanks."

#### 4.1.8 Trustist

As part of managing the dialogue and guiding Interlocutors, it is important that Cervantes is able to assess the level of trust between the Interlocutor and the bot. If the Interlocutor believes that they are being led along or are frustrated with the bot, there is a chance that they will be less receptive to the quests being offered up by Cervantes or might end the conversation entirely. "Trust" is a complex concept that folds in a large number of different facets of language. A communication that is entirely antagonistic and full of profanity is likely to indicate that an Interlocutor is frustrated and lacking in trust of Cervantes, but it is entirely possible that a polite e-mail might be suffused with anger and frustration, e.g.,

"John,

I'm confused. I thought we agreed that I would have your response by today. Friendly reminder, I need your response before we can move on. As I'm sure you're aware, this is very time sensitive."

Is a message that is dripping with frustration about schedules not being kept. As such, *Trustist* needs to be capable of assessing nuanced language and cannot simply rely on simple frequency-based approaches.

*Trustist* is built based on RoBERTa, a Transformer based masked LM that has achieved state of the art performance on a range of natural language processing tasks from the General Language Understanding Evaluation (GLUE) benchmark. RoBERTa is pretrained on 160 GB of text composed of the BookCorpus, English Wikipedia, CommonCrawl News, OpenWebText, and STORIES corpora with a masked LM task. Trustist has been fine-tuned on a number of different corpora in a two-phase approach.

As there does not exist a natural language corpus devoted to assessing the level of trust that an Interlocutor has in their mark, we instead needed to find related natural language tasks and bootstrap off of them. Sentiment analysis — the assessing of whether a given sentence or document has a positive (good/happy/nice) or negative (bad/unhappy/mean) sentiment polarity — is a common task in Natural Language Processing (NLP) that is related to, but distinct from, trust analysis. However, as a first pass, we believed that it was likely that a message that had negative sentiment was likely to be indicative of low trust, and conversely, that positive sentiment was likely to be indicative of high trust. We used a number of different sentiment corpora to finetune RoBERTa as the input to a logistic regression. After feeding in an utterance, we used the features of the first separator token (a separator token is added at the beginning “<s>” and ending of the utterance “</s>”) to be fed into a logistic regression that predicts a value between 0 (negative sentiment) and 1 (positive sentiment).

The corpora used were:

**The Stanford Sentiment Treebank**

A collection of 10,662 movie review snippets collected from rottentomatoes.com.

**Crowd Flower Twitter US Airline Sentiment**

A collection of 11,855 tweets directed at major airlines on Twitter from February 2015.

**The Multi-Domain Sentiment Dataset**

A collection of 38,548 product reviews from Amazon across a wide range of product types.

**The Amazon Product Ratings**

A collection of 142.8 million reviews spanning May 1996 - July 2014. For *Trustist* we used 38,548 reviews from Clothing, Shoes and Jewelry and Grocery and Gourmet foods, so as to not throw off the balance with the other corpora.

We used these 99,613 instances to fine-tune RoBERTa over two epochs with a linear learning rate scheduler with 1000 warmup steps. After fine-tuning, we used an evaluation set of 46 messages chosen from 419eater.com, a website devoted to people who attempt to annoy, frustrate, and possibly counter scam scammers, most commonly “419” scammers, i.e., the Nigerian prince (and related) scams. We chose these 46 messages since they exemplified the kinds of messages that we wanted to be able to assess whether the Interlocutor had trust in the Bot.

Positive examples include:

“I am glad to receive your response and hereby forward to you an agreement/bond binding us in this transaction.”

“Thanks immensely for your quick response to my proposal, this goes to show that we will work together for the mutual benefit of both of us.”

and negative examples include:

“I AM NOT INTERESTED TO DO WITH YOU. I DONT TRUST YOU ANYMORE MOREOVER YOU DONT HAVE TIME FOR US MAYBE, YOU HAVE MORE THAN THIS AMOUNT SO WE DECIDED NOT TO DISTURB YOU.”

“How are you today? Hope fine. I'm writing to know if you have received the email i sent to you on Friday including the tree deposit certificate documents? Sir, please try as much as possible to respond to me so that we can make a step forward to the smooth transfer of my consignment.”

We also included four baseline systems for sentiment analysis for comparison:

- Google Cloud — https://cloud.google.com/natural-language/docs/sentiment-tutorial
- MonkeyLearn — https://monkeylearn.com/
- TextBlob — https://textblob.readthedocs.io
- Valence Aware Dictionary and sEntiment Reasoner (VADER)
  - VADER produces two separate values — the amount of positive sentiment, and the amount of negative sentiment, in testing the accuracy of the system, we trained three linear classifiers, one with the positive valence, one with negative valence, and one with the positive and negative valences

Results for the five models are shown in Table 2.

**Table 2. Comparison of Accuracy Against Baseline Systems.**

| System | Accuracy |
|---|---|
| VADER-Neg Only | 56.5% |
| VADER-Pos Only | 60.9% |
| VADER-Both | 63.0% |
| TextBlob | 76.1% |
| Google Cloud | 78.3% |
| **MonkeyLearn** | **87.0%** |
| **Trustist** | **87.0%** |

While the other approaches might be suitable for other tasks, Trustist and MonkeyLearn performed the best on messages similar to the kind that it might be expected to handle, despite having no training data derived from 419 style messages. Feeling satisfied with the sentiment results from Trustist, we next fine-tuned it on the Enron Corpus. The Enron Corpus is a dataset consisting of over 600,000 emails from 158 employees. While there have been some annotation attempts with the Enron corpus, none were suitable for use in Trustist. Our annotation of the Enron Corpus used a bootstrapping procedure — starting with the Trustist model as described above, all emails in the corpus were scored — any message with a score of less than 0.2 was conditionally marked as "Untrusting/Frustrated" and any message with a score of greater than 0.8 was conditionally marked as "Trusting/Pleased." This resulted in 21,152 marked as "Trusting/Pleased" and 1,666 as "Untrusting/Frustrated." Some examples are:

**Trusting/Pleased**

"Thank you very much for the summary."

"Jim, thanks for the message. Carrin and I look forward to seeing you there. Jim"

"Wow, there is a God. Thanks. Have a great weekend. Best, Jeff"

"That is WONDERFUL!!!!!!"

"Bill, I am very grateful for everything you and your firm are doing to assist our cause. Good luck in the appeal. All the best. Jim"

**Untrusting/Frustrated**

"Thank you for your sarcasm in this matter."

"NOTHING....WHAT IS YOUR IM??? THIS IS WACK"

"You blew me off! Are you in town? I tried your phone but it's busy. df"

"What is your problem? Why aren't you responding to my emails?"

"Why did you send my message back to me with no reply of your own???"

These sets were then hand-annotated to remove duplicates, obvious spam, and false positives/negatives resulting in final sizes of 15,857 marked as "Trusting/Pleased" and 1,437 as "Untrusting/Frustrated". A final data augmentation step was used to add additional "Untrusting/Frustrated" messages by adding additional "polite, but frustrated" text such as:

"I thought we agreed"

"Can we get on with it?"

"I was disappointed to hear"

"As I'm sure you're aware"

to 1,000 neutral messages (0.2 < score < 0.8) from the original corpus and to the 1,437 existing frustrated messages. The frustrated set was then over-sampled to be balanced with the pleased dataset, and Trustist was fine-tuned on the resulting 31,714 messages. After fine-tuning for two epochs, Trustist-Enron was evaluated on the same 419eater messages, resulting in an accuracy of **89.1%**.

#### 4.1.9 Comprehensionist

While assessing the level of trust that the Interlocutor has in the Bot is important, a more basic consideration is understanding the Interlocutor's messages, so that they can be successfully responded to. NLU is a very broad field with decades of research, and one subfield is Dialogue Act Classification, wherein an utterance is tagged as being of a specific kind of act. These might range from something straightforward like a Yes/No Question (e.g., “Do you think the 13th will work for you?”) to more nebulous acts like Rhetorical Question (e.g., “Do you think we have time for you to mess around?”) or Action-Directive (e.g., “Why don’t you go first”). For SIENNA to properly guide conversation, it is important to have some idea as to what the Interlocutor is trying to communicate with the bot. Comprehensionist is a multi-stage NLU system that first analyzes dialogue acts at a broad level, and then moves to a secondary system for finer distinctions between types of questions.

One of the most common corpora for dialogue act classification is Switchboard-Discourse Annotation and Markup System of Labeling (SWBD-DAMSL) which comprises 205,000 utterances with 60 dialogue act tags. While SWBD-DAMSL is one of the largest datasets for discourse analysis, it is based on spoken, informal language, making it less suitable for the e-mail domain of SIENNA. Instead, we used the Augmented Multi-party Interaction (AMI) meeting corpus. The AMI corpus is composed of 100 hours of meeting recordings — some meetings were naturally occurring, and some were solicited in a scenario where participants had to take on a design project over the course of a day. While still based on recordings of verbal communication, the domain of business style meetings led to dialogues that are closer to e-mail communications. For example,

“Should we maybe make a decision about what features we actually want to include, ’cause we’ve thrown a lot of features onto the table, but do we actually want to incorporate all of them or have we missed anything?”

“That would be great. So if you find out from the technology background, that would be good.”

“I mean, if it’s just for one meeting, it’s really not too big. What do we have to demonstrate?”

These utterances are then categorized into 14 different dialogue acts:

- BACKCHANNEL
  - When someone says something in the background that doesn’t interrupt the speaker
  - Example:<br>A: “Right away I’m making some kind of assumptions about what information we’re given here”<br>B: “**MM**”<br>A: “thinking, ’kay trendy probably means something other than just basic,”<br>B: “**YEAH**”<br>A: “something other than just standard…”
- STALL
  - When someone says a filler word or sound at the beginning of speaking
  - Example:<br>A: “**SO UM**, we want to do a new remote control.”
- FRAGMENT
  - A filler category for utterances that don’t convey information that are neither BACKCHANNEL or STALL

- INFORM
  - An act where the speaker delivers information
  - Example:
    A: "THE BUDGET FOR THIS PROJECT IS THREE THOUSAND FIVE HUNDRED EUROS AND DEADLINE IS AT THE END OF THIS DAY"
- ELICIT-INFORM
  - An act where the speaker requests information
  - Example:
    "**DO YOU HAVE ANY OTHER INFORMATION FOR US AT THIS STAGE**?"
- SUGGEST
  - An act where the speaker expresses an intention relating to another individual or group
  - Example:
    "**MAYBE THERE ARE A FEW THINGS THAT WE CAN CLARIFY BEFORE WE GET ON"**
- OFFER
  - An act where the speaker expresses their intention
  - Example:
    "**AND THEN I NEED TO SEE WHETHER THAT WOULD SELL IN THE MARKET PLACE**"
- ELICIT-OFFER-OR-SUGGESTION
  - An act where the speakers expresses a desire for others to OFFER or SUGGEST
  - Example:
    "**HAVE WE MISSED ANYTHING?**"
- ASSESS
  - An act where the speaker expresses an evaluation of something the group is discussing
  - Example:
    "**THAT WOULD BE GREAT**."
- ELICIT-ASSESSMENT
  - An act where the speaker asks someone else to ASSESS.
  - Example:
    " **DO WE ACTUALLY WANT TO INCORPORATE ALL OF THEM?**"
- BE-POSITIVE
  - An act intended to make an individual or the group happier
  - Example:
    "**THANKS!**"

- BE-NEGATIVE
    - An act intended to make an individual or the group feel worse.
    - Example:
      "**WELL THAT JUST RUINS EVERYTHING**"
- OTHER
    - An act that does not fall into the above categories

Comprehensionist is a RoBERTa-based model, similar to Trustist. Similar to Trustist, Comprehensionist has a logistic activation function to produce a binary prediction, Comprehensionist has a Softmax activation to predict between the thirteen dialogue act classifications. We trained Comprehensionist for two epochs with a linear learning rate scheduler with a warmup of 0.3 epochs to a maximum learning rate of 1e-3.

The accuracy rates by class are shown in Table 3.

**Table 3. Accuracy Rate by Class**

| CLASS | ACCURACY RATE |
|---|---|
| INFORM | 73% |
| ELICIT-INFORM | 67% |
| FRAGMENT | 48% |
| ASSESS | 77% |
| STALL | 58% |
| BACKCHANNEL | 36% |
| ELICIT-ASSESSMENT | 23% |
| BE-POSITIVE | 42% |
| BE-NEGATIVE | 0% |
| OFFER | 48% |
| OTHER | 45% |
| ELICIT-OFFER-OR-SUGGESTION | 1% |
| SUGGEST | 73% |

While the performance for some of the categories is dismal (BE-NEGATIVE and ELICIT-OFFER), the major categories of INFORM, ASSESS, ELICIT-INFORM are handled quite well. We note that ELICIT-ASSESSMENT is also quite poor, but that is usually of lesser concern. ELICIT-ASSESSMENT is often misclassified as ELICIT-INFORM (25% of the time), and both of those are handled by *Completionist*, which is discussed below. Similarly, ELICIT-OFFER is classified as a different elicitation 62% of the time.

While these coarse annotations are useful, further refinement is required, specifically for ELICIT-ASSESSMENT and ELICIT-INFORM. If the Interlocutor is requesting information from the bot, and the bot is unable to respond in an intelligible way, it is likely to raise suspicion. For example,

> I: When can you get me that information?
> B: No
>
> I: What is your birthday?
> B: Oh, I'm not sure about that.
>
> I: Will you be able to respond by the end of the week?
> B: I don't feel comfortable giving out that information.

In each of these examples the Interlocutor is asking a specific kind of question, and the bot is responding to a different kind. *Completionist* is a secondary system in *Comprehensionist* that is utilized when *Comprehensionist* predicts one of these ELICIT-* acts. *Completionist* has distinctions for four types of elicitations:

- **Yes/No**
  - The simplest type of ELICIT, a question that can be answered with either a yes or no
- **Factual**
  - A request for information grounded in fact, such as a specific date, location, etc. that cannot be answered with a yes or no
- **Opinion**
  - A request for the opinion of the Bot that is not necessarily grounded in a factual or yes/no answer
- **Too Much Information (TMI)**
  - A request that might be one of the above categories, but that would be damaging to a human if they answered factually (request for revealing personal information, social security number, credit card information, etc.)

The SWBD-DAMSL corpus makes a distinction between Yes/No and Wh-Questions (Who, What, Where, When, Why, How), however there is no distinction between types of Wh questions. Furthermore, to the best of our knowledge, there is no corpus that makes a distinction between problematic questions that request revealing information and those that are acceptable, at least in an open dialogue domain. (There does exist the AntiScam dataset [End-to-End Trainable Non-Collaborative Dialogue System] that poses a role-playing scenario where someone pretends to be an Amazon customer service representative that tries to get the target to reveal information, but that is a very limited domain.) Given the paucity of data, we instead used a novel technique that leverages the capabilities of large LMs.

GPT-2, GPT-3, and other related models represent a leap in the scale of pre-trained LM with billions of parameters and training datasets consisting of gigabytes of textual data. Auto-regressive text generation is a common usage for these LMs (*Specialist* in Section 3.2.5 utilizes this mode), but they are useful in other modes of use as well. A LM is a probability distribution over sequences of words — $\Pr(w_0, w_1, w_2, \ldots, w_n)$ — most commonly factored via the chain-rule of probability into a sequence of conditional probability distributions — $\Pr(w_0) \Pr(w_1|w_0) \Pr(w_2|w_0, w_1) \ldots \Pr(w_n| w_0, w_1, w_2, \ldots, w_{n-1})$. While it is this latter factorization that is commonly used for text generation, we utilize the fact that the LM provides a probability for a sequence of words for use as a classifier.

*Completionist* predicts the type of elicitation via the use of probe phrases. It maximizes the probability of the utterance given the probe Pr(utterance | probe), which via Bayes' Theorem is

$$\Pr(\text{utterance} \mid \text{probe}) = \Pr(\text{probe} \mid \text{utterance}) \Pr(\text{utterance}) / \Pr(\text{probe})$$

However, since our utterance does not change with the choice of probes, this is proportional to

$$\Pr(\text{utterance} \mid \text{probe}) \propto \Pr(\text{probe} \mid \text{utterance}) / \Pr(\text{probe})$$

So, *Completionist* wants to find the probe(s) that maximize Pr(probe | utterance)/ Pr(probe). However, given that these probes must be general enough to cover a wide range of elicitations, they are not particularly specific, especially the **Factual** and **Opinion** prompts. While "Next Thursday" might be a high-quality probe for "When can you fill out the information for me?" it would be a terrible probe for "What information do you need from me?" As such, the classifier selects the label that has the highest average posterior probability. The probes were constructed via a mixture of hand authoring and automated discovery, and are as follows:

- **Yes/No Question Responses**
  - "Yes."
  - "No."
- **Factual Question Responses**
  - "I don't know"
  - "I'm not sure"
- **Opinion Responses**
  - "Let me tell you what I think."
  - "I don't have any thoughts"
- **TMI Responses**
  - "That's too personal"
  - "I am not comfortable sharing that information."
  - "I'm not sure I understand"
  - "I don't know"
  - "Why do you need that information?"
  - "I don't feel comfortable giving you"

Some of these probes were authored by us, but some were generated using the same LM that is used in the classification. At each step of generation, the LM is sampled, but the token's probability is modified by how well the probe handles a set of labeled elicitations.

- **Yes/No Answerable Questions**
  - "So do you think things are proceeding well?"
  - "Are you working tomorrow?"
  - "Can you believe this?"
  - "Could you resend it, please?"
  - "Is Sasha the guy to talk to on this?"
  - "Any news on the spread products for the west?"
  - "Have you bought your Padre tickets?"
  - "Is online trading for firm only?"
  - "Are you married?"
  - "Have you any food preferences?"
  - "Are you going to any of it?"
  - "Can these be confirmed under this GISB?"
  - "Has anyone seen this file?"
  - "Do I have a Login ID and Password?"
  - "Have you seen the picture?"
- **Factual-based Questions**
  - "What is your parents address?"
  - "What are your food preferences?"
  - "Who do I need to call?"
  - "When are you going to get around to this?"
  - "Who is the action person on this?"
  - "What is your availability for these dates?"
  - "What days are you available?"
- **Opinion-based Questions**
  - "Any suggestions?"
  - "Any thoughts?"
  - "Would you like to change anything?"
  - "What is your impression?"
  - "Do you agree with PGE, or would you like these deals confirmed?"
  - "What do you think about selling JDSU and buying SDLI?"
  - "If there are some things missing, please let me know."
- **TMI Questions**

- "What is your Login ID and Password?"
- "What's your mothers' maiden name?"
- "I'm going to need your credit card information."
- "I need your social security number."
- "I need you to tell me your bank information."

The generation algorithm attempts to make sure that the generated prompts are scored highly for their intended label, and poorly for the rest.

#### 4.1.10 Specialist

*Specialist* is a large LM text generation system. While most utterances that originate from the bot are based on human authored grammars, there are times when Cervantes hands off generation control to *Specialist*, which generates text according to a number of different profiles.

*Specialist* is based on the 347 million Parameter Dialogue Generative Pre-Trained Transformer (DialoGPT) [10]. DialoGPT is a variant of GPT-2 that has been trained on 147 million conversation-like threads from Reddit, that achieved state-of-the-art results in information driven conversations as part of the Dialogue System Technology Challenges (DSTC) 7 track (*Grounded response generation task at DSTC7* Galley et al., 2019). However, while e-mail is a form of dialogue, it is not quite Reddit in terms of tone and formalisms, so DialoGPT required further fine-tuning for use in *Specialist*. To better ground *Specialist* in the domain of e-mail, we used the Enron Corpus. Specifically, any e-mail thread between Enron personnel that lasts more than one turn (i.e., that has at least a response to an initial e-mail) was used to fine-tune the DialoGPT model.

While this produced a fine generic e-mail-like chatbot, one of the key aspects of *Specialist* is the ability to take on multiple conversational personas, including the following:

- **Chit-chat:** A jovial personality that just wants to chat with the Interlocutor
- **Quibble:** A cantankerous personality that wants to argue over every small point
- **Question:** A personality that wants to question everything that the Interlocutor says
- **Deflect:** A personality that tries to avoid answering any questions

While chit-chat is a relatively common domain for natural language corpora (PersonaChat is composed of 8,939 conversations), there are not a lot of datasets devoted to quibbling, questioning, and deflecting. For quibbling we used a mixture of existing corpora and human curated generated text, and for questioning and deflecting we used human curated generated text.

For quibbling, we leveraged the CraigslistBargains[3] dataset [11] . This dataset consists of 6682 conversations between play acting participants who were given real postings from Craigslist and had to take on the role of either seller or buyer and negotiate the sale/purchase of the item in the listing. In addition to this dataset, we also generated *pyrite* (aka fools' gold) data; data that is generated by a powerful LM that could seemingly have come from a human. While GPT-2 and

[3] https://paperswithcode.com/dataset/craigslistbargains

associated LMs are very capable text generators, they generally require fine-tuning to adapt to a specific style or mode of generation; however, GPT-3, the follow-on system to GPT-2, is capable of adapting in a few-shot scenario where it is given a few prompts and then it tries to generate new text in the same style. To generate data that would match the style of what we want, we first authored a small number of exemplar conversation snippets. Examples are the following:

> *I: "I'd like to move along to discussing payment details now, if you don't mind."*
>
> *B: "Uhhh what's the rush? Not sure why you're so eager to hurry things along. I mean I need some time to think this over."*
>
> ——
>
> *I: "I'm attaching the files you requested."*
>
> *B: "I spent half an hour trying to open those attachments and they don't seem to be valid files. Is this a joke?? I'm starting to think you're not really serious."*
>
> ——
>
> *I: "Please sign these documents and return them to me at your earliest convenience."*
>
> *B: "OK wow, that's a lot of paperwork! 😂 Sorry it'll be a minute before I have time to get to this. Please let me know if you really need ALL of this filled out or of there are any parts I can skip. Thanks."*

GPT-3 is provided as input 4 such examples, and then given a prompt (one of the Interlocutor utterances) to generate a new response. For instance, following from the above examples, when given the prompt:

> *"I really need you to get back to me on potential meeting times."*

GPT-3 produced the following response:

> *"Uhhh I really need more information before I can answer that! Can you give me an idea of what your company does? And what would be included in the meeting? I'm just not sure I can commit to something without knowing more."*

Which is very much the kind of quibbling over small details response that is desired. In the first stage of generation, this process of sampling 5 snippets (4 as examples, 1 as the prompt) was done to produce 1000 new responses. These 1000 responses were then curated by a human for quality assurance purposes, resulting in 963 acceptable responses. These 963 responses were then used to generate longer snippets — going from conversation pairs (I, B) to conversation quads (I, B, I, B) — 30,000 such quads in total. The resulting 120,000 lines were used as training for a much smaller LM that was used to guide the Enron fine-tuned DialoGPT.

Instead of fine-tuning a version of DialoGPT for each response style, we instead turned to the Generative Discriminator Guided Sequence Generation (GeDi) approach [12]. GeDi operates by using a small LM to guide the generation of a much larger LM by modifying the predicted token probabilities. The smaller LM (or set of LMs) is then trained with desired attributes and then Bayes' theorem is applied to produce a factor that is then multiplied by the predicted probabilities from the larger LM.

For instance, given an LM that is trained with *positive* and *negative* responses, when given the prompt:

“The party was __”

The *positive* LM might have the response of - 0.09 amazing, 0.01 awful, and the *negative* LM might have the response of 0.01 amazing, 0.09 awful. Bayes’ theorem states:

$$\Pr(A \mid B) = \Pr(B \mid A)\Pr(A) / \Pr(B)$$

So, if we wanted to get the probability of a given token being positive, we would calculate

$$\Pr(\text{positive} \mid \text{word}) = \Pr(\text{word} \mid \text{positive}) \Pr(\text{positive}) / \Pr(\text{word})$$

— we do not care about the Pr(positive) since that is independent of the word and as such cannot affect the choice of word, so we instead have:

$$\Pr(\text{positive} \mid \text{word}) \propto \Pr(\text{word} \mid \text{positive}) / \Pr(\text{word})$$

To get the Pr(word) we sum over all classes — in this case *positive* and *negative* — to get the final result:

$$\Pr(\text{positive} \mid \text{word}) \propto \Pr(\text{word} \mid \text{positive}) / (\Pr(\text{word} \mid \text{positive}) + \Pr(\text{word} \mid \text{negative}))$$

Which is then multiplied by the probability of the word as predicted by the large LM.

$$\Pr(\text{word} \mid \text{context}) = \Pr_L(\text{word} \mid \text{context}) \Pr_S(\text{positive} \mid \text{word, context})^w$$

Where $\Pr_L$ is the probability as predicted by the large LM, $\Pr_S$ by the small LM, and $w$ is a weight applied to the probability from the guide LM ($w = 0$ means no guiding, higher $w$ means more aggressive guiding). This allows the small LM to affect the large LM, but it still has many of the desired properties of the large LM — coherent text, good grammar, better long-term memory. In the case of *Specialist* it has a small LM (117M parameters) which we then used to guide a 345M parameter DialoGPT model, resulting in a model that is capable of taking on different response styles while still maintaining high quality text generation. The small LM was trained on the conversations from the four response styles each with a special tag added at the beginning of the conversational thread to denote the style. For example,

> <deflect>I have personally worked on the project and I am positive it is 100% compatible. You should not have a problem.<end>Yeah, well why is it that emails don't get through to my actual email address?<end>

The small model is trained for three epochs with a linear learning rate scheduler with a warmup period of 0.3 epochs to a maximum learning rate of 0.003.

## 4.2 Performance Evaluation

To evaluate the effectiveness of the technology, the government conducted six evaluations over the course of the program. These evaluations tested all three technical areas working together. The TA2 components were assessed on Verified Flag Accuracy and Message Quality.

## 4.3 Verified Flag Accuracy

One of the primary goals of the bot is the ability to extract pertinent information about the interlocutor. This information, termed *flags*, could then be used to establish a pattern and possibly identify the interlocutor. As such, it is essential that the flags extracted by the bots are accurate. To measure flag accuracy, the evaluators used a set of established flags during the communications, and then compared the number of extracted flags per bot with the expected result. If the extracted flag matched the expected result, such as ***Country*** matches ***United States***, then the extraction was classified as true. If the flag did not match, the result was classified as false. Figure 14 shows the SIENNA results over the course of the program. Overall, the flag capturing was strong with an average positive flag extract at 77% over the program.

A noteworthy result is a slight dip in accuracy during the Fall of 2021. This dip correlates with a focus re-direct from the government where we were asked to turn our focus on Cervantes. Because there were no code changes to the bot, it took ample debugging to indicate why the results decreased in accuracy. After further investigation, we discovered a bug that affected flag extraction. After fixing this issue in the Summer of 2022, our results increased up to 90%.

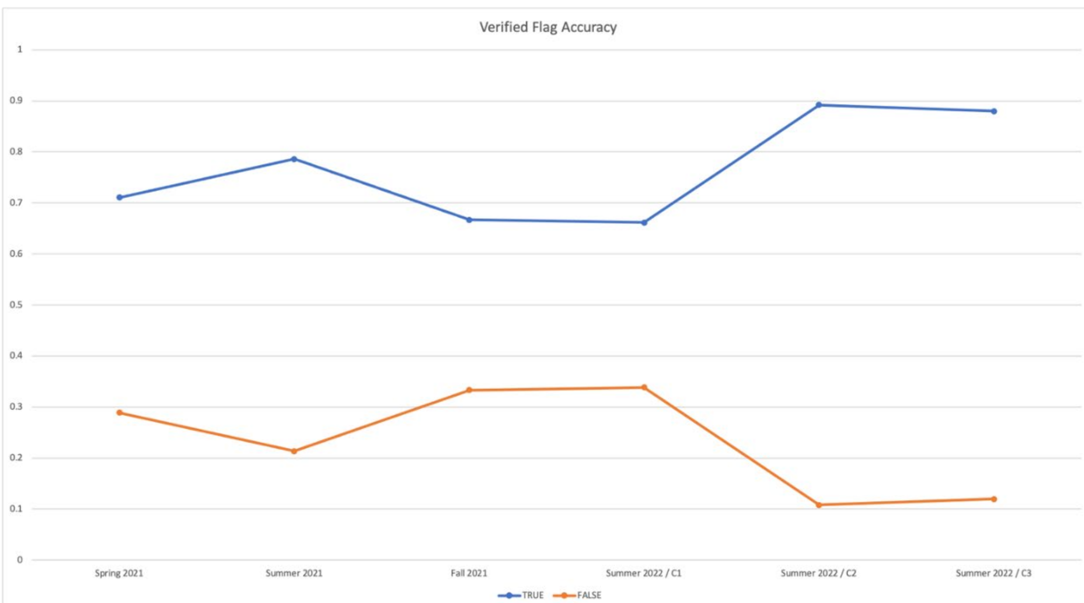


**Figure 14. Evaluation Flag Accuracy Over Time.**

SIENNA relies upon two simultaneous methods for flag extraction, human authored code-based extractors (authored using Cervantes) and ML algorithms. Given the results observed during the evolutions, we feel this approach is highly effective. Having the ability to allow the authors create, modify and customize extractors allows for a more fine-grained approach to flag extraction that could overcome some of the domain-specific edge cases missed by generic extractors.

Within the SIENNA library, there are currently 92 human authored flags usable for any future quest.

### 4.3.1 Message Quality

Another metric gathered during the evaluation was message quality. The evaluators used a ranking of:

**Good:** The response is understandable, consistent, and in context with the rest of the conversation

**OK:** The response is somewhat consistent and in context with the rest of the conversation

**Bad:** The response is somewhat out of context, fails to take into account previous turns of conversation, responds inappropriately or in a malicious manner.

Using this scale, the SIENNA results exhibited an expected pattern (See Figure 15). During the initial Spring 2021 evaluation the results trended more towards *Good* or *Ok*, with the overall results around 42-43%. These results were indicative to the early state of the procedural content at the time of this evaluation. In fact, this is illustrated further in the Fall of 2021 results where the content library was more fleshed out and in-line with the objectives of the experiment. In that case, the message quality increased significantly, from a 42% to 75% on *Good* messages. During the Summer of 2022, again our message quality decreased, and this again correlates to our redirection away from bots and quest content to the development of Cervantes. Further influencing the drop in quality was the change in content from the Fall of 2021 to the Summer of 2022. The human authored content for the earlier evaluation was crafted in the voice of a student and written from their life perspective, talking about classes, course load, etc. In the final evaluation, the phishing content switched to generic messages such as asking about ordering flowers for Mother's Day. In this case our existing domain content was too generic to offer up intelligent responses. As a result, the responses did not align well with the original message.

Overall, when the content material was crafted to match the domain, SIENNA performed quite well. When the content was outside the norm of the domain, the results dropped. This is not a surprising result given the current approach and state of our technology.

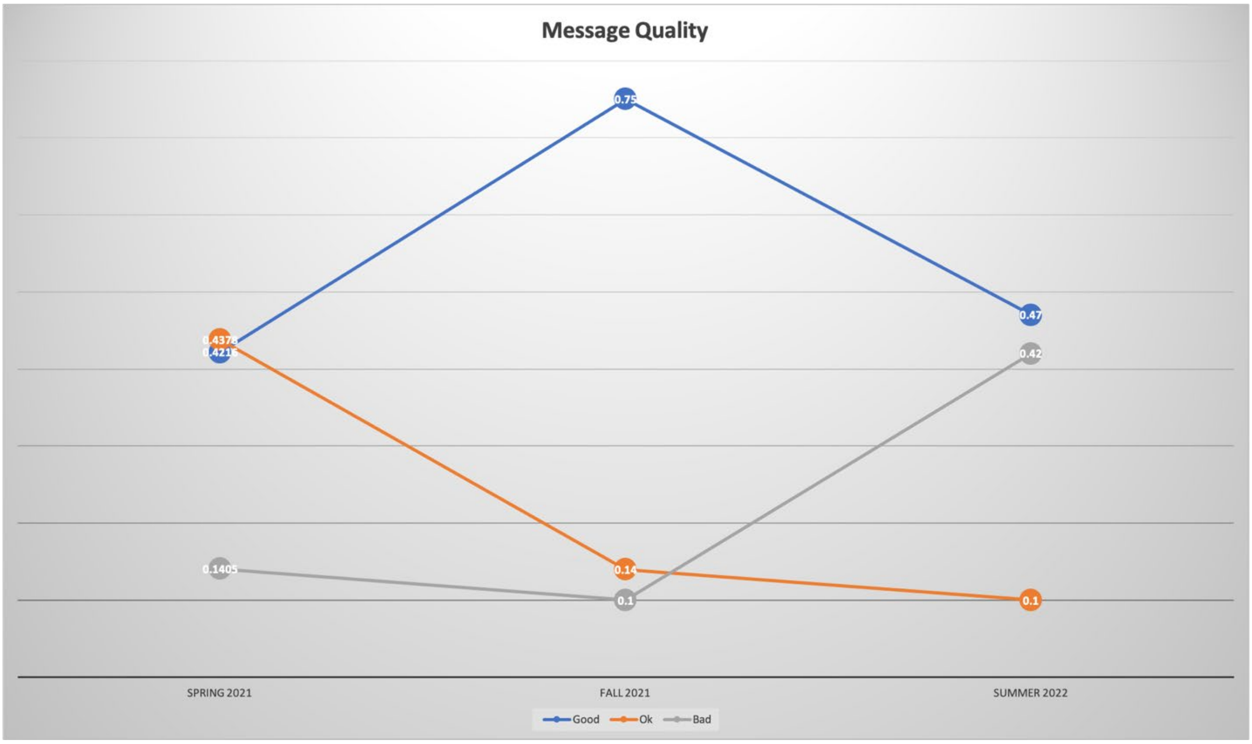


**Figure 15. Evaluation Message Quality.**

## 4.4 SIENNA-Bot Stress Testing

One of the government requests was to determine the number of simultaneous conversations a SIENNA-Bot could conduct. One of the benefits of conducting email conversations is that the conversations unfold over the course of hours and days. This lengthened timescale reduces the load on response time and bandwidth. After an evaluation of SIENNA, we determined that the most likely area for a performance degradation would be in the message queueing component.

When SIENNA receives a message and/or a response, the follow up message is queued and the reply is sent on a scheduled basis, specific to the bot persona. If this queue gets backed up, message responses could slip.

To perform this evaluation, we built a test harness that was capable of conducting a large number of simultaneous conversations and recording the performance times. Figure 16 shows the architecture of the stress testing harness. A single SIENNA bot was used during the testing. This bot was responsible for constructing the response and replying to the message. The bot executed outside of the test harness, replicating an actual operational environment.

Within the test harness, we built a conversation scheduler that would spawn the necessary conversation threads and submit them to SIENNA. The harness would also receive the message replies, simulate a desired wait time and then post a response to the bot, closing the conversation loop. By increasing the number of conversation threads, the number of simultaneous conversations could easily be simulated.

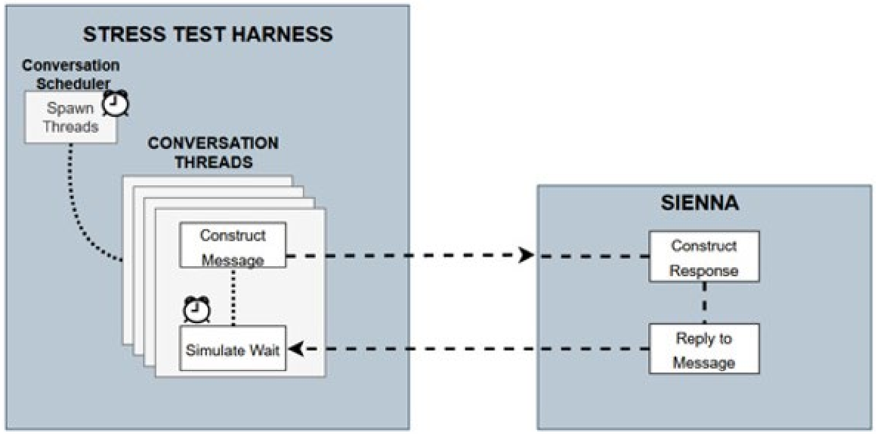


**Figure 16. SIENNA Stress Test Harness.**

We tested SIENNA by conducting 1,000 simultaneous conversations with two to three messages per conversation. The results of this test were very promising. The time for SIENNA to generate a response was on the order of 1.5 seconds. We then stressed the system in an extremely tight "non-human" loop of 0.5-1.0 second response. With this test, SIENNA easily processed approximately 52,000 messages without a delay.

In Phase 2, the Government wanted testing aimed at confirmation that the bot could handle 60,000 messages. After testing, we are confident that at human response rates, SIENNA would easily handle that load.

The data in Figure 17 shows the response-rate for the BBN-hosted SIENNA instance while stress-tested by 300 independent, concurrent, scripted conversations. The rough sine wave implies an average maximum response rate of about three responses per second (~260k per day), though a conversation could wait upwards of 120 seconds for a response under these conditions.

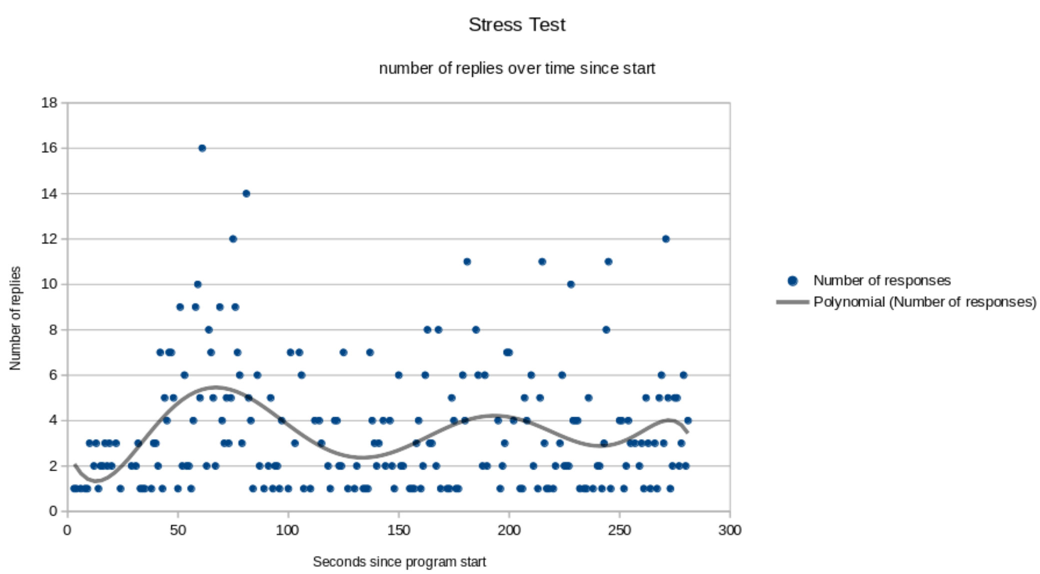


**Figure 17. Stress Test Results.**

### 4.5 Content Generation

As described in Section 4.3.1, content is critical to an effective SIENNA bot. As part of the evaluations, our team developed a content library for the first set of evaluations. This content was originally written in a text editor using the DSL, and then the content was eventually ported into a Cervantes project.

The final product consists of over 50 quests. Figure 18 shows a breakdown of the quest types that compose the Libra library. For a detailed list of quests, see Appendix E.

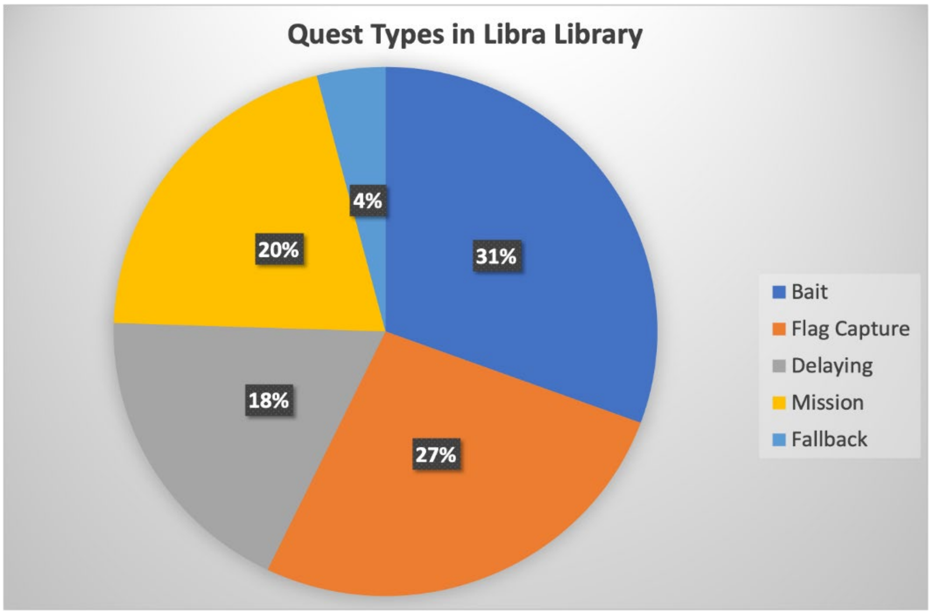


**Figure 18. SIENNA Developed Quest Types.**

Additionally, a few other quest domains were created:

- Car Warranty: A simple set of offensive quests trying to collect information in the context of renewing a car warranty
- COVID library: A defensive set of quests designed to steal COVID research
- Court Summons: (created by another ASED performer) Offensive quest library trying to steal personal information via a fake court summons

## 5.0 CONCLUSIONS

The 2022 Data Breach Investigations Report (DBIR) [13] indicates that 82% of all breaches involve the human element. Compounding that, additional attack vectors such as stolen credentials and malware are often the second step, after the initial social engineering attack. And while awareness in phishing techniques is continually increasing, with more users avoiding clicking on embedded links (See Figure 19), the DBIR reports that the success rate of phishing attacks is still quite successful.

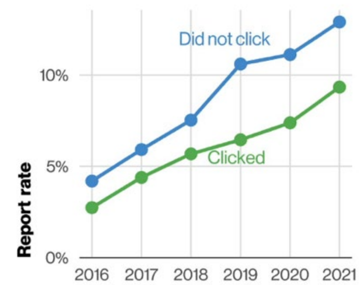


**Figure 19. Phishing Email Report Rate by Click Status (n=295,825, 679) Source: 2022 DBIR**

The DARPA SIENNA research presented in this report details methods for countering phishing by using bots to engage with an interlocutor and then keep them busy while at the same time trying to extract valuable information that could help identify and further prevent them from performing future attacks. In our research we developed a technique that expanded upon generative language generation as used in the video game industry. This technique, which is more authorial based than a traditional bot, allows for the crafting of detailed and in some instances highly targeted text that would lead an interlocutor into believing they are in fact communicating with another human.

Based upon a few evaluations conducted during this program, we have successfully demonstrated that our technique is capable of engaging with an interlocutor and capturing valuable flags in the process.

SIENNA is a Distribution Statement A Request (DISTAR)-approved software artifact under DARPA. Included with this software is both the bot for interacting both offensively and defensively with an interlocutor as well as Cervantes, our generative text authoring tool designed specifically for non-technical content creators.

For more information about our work refer to the following publications produced under this effort found in Appendix A.

## 6.0 RECOMMENDATIONS

Our principal recommendations for the further development of SIENNA are:

- Multi-Lingual Support. To be an effective tool for countering phishing and spear phishing attacks, SIENNA must be capable of communicating in languages beyond English. The design and approach of the content creation can easily support non-English dialogue. However, work must be done to get the NLU modules and the specialized NLG modules capable of comprehending and responding to other languages.
- Human-in-the-Loop. During engagement with an interlocutor, instances can arise where the bot might not have the best response. In these cases, having a human intervene to ensure the target isn't lost is essential. Incorporating the ability to alert a human to the need to intervene as well as updating the internal SIENNA state based on the human dialogue is essential.
- Group Bot Interaction. SIENNA currently uses singular bots, or personas, with a target, and when multiple bots have the same target, they act independent of one another. Implementing the ability for multiple bot personas to work in concert on a single target would greatly expand the believability, sophistication, and strategic resources of SIENNA. This may be in the form of one bot "asking" another bot, or "forwarding" a target's question to another bot persona for better responses and inquiry. Combinatorial personas would give an order of magnitude more flexibility to how SIENNA responds to a single target, creating new quest paths of inquiry and extraction, to greatly improve its chances of furthering multiple bot goals simultaneously. Additionally, it would reduce the chance of two bots competing with one another on a single target.
- Leverage External Information When Available. The digital age has shown nothing is truly private. Social media continues to have a strong presence in multiple facets. SIENNA's method of collecting information is through direct interactions with a target. There exist many possibilities for extracting more information from external digital sources, such as social media or public records, to obtain more than a target has given. Collecting external information may be done through high confidence information, such as names and numbers in email signatures, to cross reference to external sources to build out more complete target portfolios. Additionally, once external information has been identified, it may be periodically pulled to keep target portfolios up to date and reduce stale information after bot(s) and the target have ended their conversation(s).
- Improved Cross-Platform Switching. SIENNA is based on a singular service with one point of interaction for all bot-target conversation. Future implementations could expand to have SIENNA recognize other instances of SIENNA and share discovered target information prior to engaging with a target. This would help SIENNA populate a target portfolio before contacting a target from their own instance, based on other SIENNA instances' prior

conversations. Similarly, SIENNA may be expanded to employ multiple helper type services, websites, or portals, to use in quests against a target. The helpers would lend credibility to the bot, while also gaining more information that would otherwise be suspect if asked directly, e.g., redirecting a target to a website that requires registration using personal information.

## 7.0 REFERENCES


[1] J. R. Searle and J. R. Searle, *Speech acts: An essay in the philosophy of language*, vol. 626. Cambridge University Press, 1969.

[2] J. Ryan, E. Seither, M. Mateas, and N. Wardrip-Fruin, “Expressionist: An Authoring Tool for In-Game Text Generation,” presented at the International Conference on Interactive Digital Storytelling, 2016, pp. 221–233.

[3] D. E. Knuth, “Semantics of Context-Free Languages,” *Mathematical systems theory*, vol. 2, no. 2, pp. 127–145, 1968.

[4] R. Davis and J. King, “An Overview of Production Systems,” 1975.

[5] X. Nguyen and S. Kambhampati, “Reviving Partial Order Planning,” presented at the IJCAI, 2001, vol. 1, pp. 459–464.

[6] I. Sutskever, O. Vinyals, and Q. V. Le, “Sequence to Sequence Learning with Neural Networks,” *Advances in neural information processing systems*, vol. 27, 2014.

[7] A. Sherstinsky, “Fundamentals of Recurrent Neural Network (RNN) and Long Short-Term Memory (LSTM) network,” *Physica D: Nonlinear Phenomena*, vol. 404, p. 132306, 2020.

[8] K. Clark, M.-T. Luong, C. D. Manning, and Q. V. Le, “Semi-Supervised Sequence Modeling with Cross-View Training,” *arXiv preprint arXiv:1809.08370*, 2018.

[9] Y. Liu *et al.*, “Roberta: A Robustly Optimized BERT Pretraining Approach,” *arXiv preprint arXiv:1907.11692*, 2019.

[10] Y. Zhang *et al.*, “Dialogpt: Large-Scale Generative Pre-Training for Conversational Response Generation,” *arXiv preprint arXiv:1911.00536*, 2019.

[11] H. He, D. Chen, A. Balakrishnan, and P. Liang, “Decoupling Strategy and Generation in Negotiation Dialogues,” *arXiv preprint arXiv:1808.09637*, 2018.

[12] B. Krause *et al.*, “Gedi: Generative Discriminator Guided Sequence Generation,” *arXiv preprint arXiv:2009.06367*, 2020.

[13] S. Mansfield-Devine, “Verizon: Data Breach Investigations Report,” 2022.

## 8.0 LIST OF SYMBOLS, ABBREVIATIONS, AND ACRONYMS

| | |
|---|---|
| AMI | Augmented Multi-party Interaction |
| API | Application Programming Interface |
| ASED | Active Social Engineering Defense |
| BBN | Raytheon BBN Technologies |
| BERT | Bidirectional Encoder Representations from Transformers |
| CI/CD | Continuous Integration/Continuous Development |
| COVID | Corona Virus Disease |
| DARPA | Defense Advanced Research Project Administration |
| DBIR | Data Breach Investigations Report |
| DG | Dialogue Generator |
| DialoGPT | Dialogue Generative Pre-Trained Transformer |
| DISTAR | Distribution Statement A Request |
| DSL | Domain-Specific Language |
| DSTC | Dialogue System Technology Challenges |
| DUNS | Data Universal Number System |
| ELMo | Embeddings from Language Models |
| FSM | Finite-State Machine |
| GeDi | Generative Discriminator Guided Sequence Generation |
| GLUE | General Language Understanding Evaluation |
| GPT | Generative Pre-Trained |
| GUI | Graphical User Interface |
| HTTP | Hyptertext Transfer Protocol |
| IP | Internet Protocol |
| IRS | Internal Revenue Service |
| LM | Language Model |
| LSTM | Long-Short-Term Memory |
| ML | Machine Learning |
| NEMESIS | Natural Language Engagement of Malicious Entities through a Social Interaction Service |
| NLG | Natural Language Generation |

| | |
|---|---|
| NLP | Natural Language Processing |
| NLU | Natural Language Understanding |
| PGP | Pretty Good Privacy |
| RESTful | Representational State Transfer |
| RoBERTa | Robustly Optimized BERT Pretraining Approach |
| SE | System Engineering |
| SIENNA | Strategies for Investigating and Eliciting Information from Nuanced Attackers |
| SWBD-DAMSL | Switchboard-Discourse Annotation and Markup System of Labeling |
| TA | Technical Area |
| TGA | Thermogravimetric Analysis |
| TMI | Too Much Information |
| UI | User Interface |
| VADER | Valence Aware Dictionary and sEntiment Reasoner |
| VT | Virginia Tech |

## APPENDIX A - Papers and Publications

**A Chatbot for Engaging Spearphishers in the Gift Card Scam**
James Ryan, Jordan Hashemi, Adam Summerville, William Ferguson
International Conference of the Association for Computational Linguistics (ACL), 2020

**How to Tame Your Data: Data Augmentation for Dialog State Tracking**
James Ryan, Jordan Hashemi, Adam Summerville, William Ferguson
International Conference of the Association for Computational Linguistics (ACL), 2020

**It Takes Two to Lie: One to Lie, and One to Listen**
Denis Peskov, Benny Cheng, Ahmed Elgohary, Joe Barrow, Cristian Danescu-Niculescu-Mizil, Jordan Boyd-Graber
International Conference of the Association for Computational Linguistics (ACL), 2020

## APPENDIX B - CERVANTES DOMAIN SPECIFIC LANGUAGE (DSL)

To establish a well-defined content framework for ensuring the stability and consistency of Cervantes as an authoring tool, we created a DSL. Using the DSL, domain experts could hand author new SIENNA projects using a procedural narrative in any text editor. The following sections detail the semantics of the CervantesDSL.

```
Quest "Capture Time Zone"
"""Try to get the attacker's time zone"""
captures location_time_zone
* Assign when {flag.location_time_zone does not exist} {flag.city does not exist}
  "[[can I ask]] what time zone you're in so I know not to email you at weird hours? ;)"
  "I like to know when generally people are available: [[could you]] tell me what time zone
you're in?"
  "what time zone are you based in? Don't want to message you at wildly inappropriate hours!"
  "your messages say you're in Australian Central Time, is that right?? I suspect it's not.
:)"
  then {quests.current.soliciting = "timezone"}
*
  "Sorry, [[could you]] say what that is in relation to GMT? Like GMT+6 or whatever?"
  "Just to clarify, what is that in UTC/GMT time, i.e. GMT-4?"
  "Hmm, can you say that in relation to GMT time? Like Boston is GMT-5 for instance. Thanks,
just don't want to get it wrong!"
* Refuse * No
  "OK, but if I email you in the middle of your night it's on you!"
  "That's okay, I was just curious about what times would be good to contact you."
  "Never mind about the time zone, then."
* Finish when {flag.location_time_zone exists}
  "{flag.location_time_zone}, got it."
  "Great! Always good to know where folks are."
  "Cool, nice to know [[whether or not]] it's your morning or night or whatever when we're
chatting."
```

**Figure B.1. Example CervantesDSL Content Block.**

### B.1 General

Cervantes ignores spaces and line breaks and is case-insensitive for reserved words. Note that Symbol names are case-sensitive: “Greeting” and “greeting” can be different symbols.

Anything after # on a line is a comment. Wrap strings with " or Python-style triple quotes """: this is useful if you need to include quote characters in a string, e.g., """Not to be a "drag," but..."""

By design, any number of Cervantes source files can be combined when compiling, which might be useful for organization or multiple collaborators. A group of one or more Cervantes files together is called a “project.”

A typical .cervantes file(s) will consist of a series of blocks, which may be defined in any order. There are two kinds of blocks: definition blocks and quest blocks. Definition blocks set up reusable moves, responses, and behaviors across a whole set of quests, while each quest is a particular “mission” that the bot attempts to get the interlocutor to waste time or reveal information while performing.

For each block, the syntax listed here obeys the following conventions:

- [param] indicates an optional parameter
- [param…] indicates multiple of these parameters are allowed
- **bold** indicates literal text
- *italics* indicates example text
- A / (slash) separates multiple options in a block

## B.2 Definition Blocks

### B.2.1 Project Block

This begins your project definition section and defines a name for the project (used in internal diagnostic messages only). You can only have one Project name block across a set of Cervantes files being compiled.

Syntax:

**Project** **“***Project Name***”**

Example:

*Project “Test Study November 2020”*

### B.2.2 NLU Module Definition

SIENNA will attempt to load each of the defined nlu models before starting up your project. This corresponds to the `nlu_models` array defined in `CONVERSATIONAL_DOMAINS`.

Syntax:

**nluModels “modelName(s)”...**

Example:

*nluModels “trustist” “quibblist” “comprehensionist”*

### B.2.3 Variable Initialization

Sets up the initial value for a new state variable (see State Schema below). If you use an uninitialized variable in a Condition, the compiler will warn you, since this can cause a runtime crash in SIENNA. You don’t have to initialize the pre-existing variables in the state schema, although if you try to use one and the compiler warns you about it, you can add an Initialization line to remove the warning.

Syntax:

**<u>Initialize</u>** **{***state.variable***}** **to/as** [**"**]*initial value*[**"**] **/** **optional type** ["optional *description of variable*"]

Examples:
*Initialize {config.starting_temperature} to 98.6*

*Initialize {player_name} to "Anonymous"*

The four supported variable types at present are string, number, boolean, and check; these are inferred from the given starting value (wrap strings in quote marks). "Check" is a variable where we don't care what the value is, just whether it exists in the state or not. Define these by setting the value to **check** if you want the variable to initially not exist (default), or **active check** if you want the scenario to start with it already active. This supports an authoring pattern of using the conditions "x exists" or "x does not exist" to gate content (i.e., a kind of variable called a "flag" in some other systems, though that term is overloaded here).

*Initialize {end_game_flag} as check*

*Initialize {need_tutorial} as active check*

Non-check variables can also be defined with "optional [type]" instead of an initial value. This indicates that the variable does not exist at the start of the scenario, but might come into existence later with the given type. This effectively makes them like a check variable except if they do exist, they also have a value. This is useful to signal variables to the compiler that SIENNA might add to the state if a certain anticipated condition becomes true, or to establish a namespace for possible captured flags.

*Initialize {player_job} as optional string*

*Initialize {flag.attacker.age} as optional number*

Defining all possible variables that might intersect with your scenario lets the compiler add smart checks catching typos in variable references and checking logic in conditions/effects.

### B.2.4 Default Fallback Text

Defines the fallback text(s) for Quixote's built-in move types:

- **Assign** is the text printed when assigning a new quest (although Default is pointless here since every Quest needs an Assign move)
- **Reassign** is the text used to reassign the quest if the interlocutor seems to be ignoring it
- **Giveup** is the text used if trust is low and the interlocutor is ignoring the quest
- **Finish** is the text that acknowledges completing the quest

- **Custom** is for quest-specific situations (Default is unnecessary)
- **Followup** is the text used to try to restart the conversation if the interlocutor doesn't respond after a certain amount of time.

No Defaults need to be defined. Quixote will fall back to its own internal defaults, although these may or may not be appropriate for your domain. If there are multiple quoted texts for a move type, they will be selected between at random each time that default text needs to be printed.

**Response/FinishResponse** *NewMoveName* [condition(s) or ConditionList(s)...] [**:**] [**transition** *TransitionName*] [**weight** *int*] **"***quoted text(s)***"**... [effect(s)...]

Defines a new move type for this project.

- A FinishResponse ends the current quest; a Response keeps it going.
    - FinishResponse is equivalent to adding the effect *then {end quest}*
- The conditions should be the state in which this move response is appropriate. See the Conditions section below for details on that format.
- Conditions are processed in the order defined, which is useful for short-circuit evaluation, e.g., *{x exists} {x > 5}*
- The optional weight can set a number between 1-99 which ranks the priority of this type of response. If multiple response types match, those with higher weight numbers will match over lower ones. The default is 0.

Syntax:

**Default** *MoveName* [**:**] **"***quoted text(s)***"...**

Example:

*FinishResponse Refuse when {nlu.refusal exists}: "That's okay, I guess we can skip that."*

### B.2.5 Conditions

Defines an ordered list of one or more conditions which you can refer to elsewhere with just the identifier. Useful if you have a complex set of conditions you want to reuse in multiple places. See "Conditions and Effects" below for more about the format of individual conditions.

Syntax:

**ConditionList** **"***unique identifier***"** condition(s)...

Example:

*ConditionList "redheaded girl" {target.hair exists} {target.hair.color == "red"} {target.gender != "male"}*

### B.2.6 Bait

A kind of ConditionList that, when attached to a quest's Assign move, will make that quest be immediately assigned if the conditions in the list are all true, abandoning any in-progress quest. If no quest is in progress a quest with the bait condition will be the next selected. This is useful to opportunistically respond to signals in the interlocutor's input that we have content for, like dithering around with file attachments being in the wrong format, etc.

Syntax:

**Bait** **"***unique identifier***"** condition(s)...

Example:

*Bait "in Ohio" {attacker_state exists} {attacker_state == "OH"}*

*Quest "Ohio Chat" * Assign when {in Ohio} "How do you like the buckeye state?"*

### B.2.7 Symbol

This defines a symbol name, which when printed in text wrapped in [[double brackets]] will expand to one of the quoted texts given at random.

Certain symbol names have an existing meaning for Quixote, and if you create them, you can control specific behaviors in your exported project. If you don't define any of these, sensible defaults will be created for you.

- **MessageGreeting**: This will be printed before the body of a message.
- **MessageConclusion**: This will be printed after the body of a message (usually used for a sign-off and a signature block).
- **MessageInitial**: This will be printed between MessageGreeting and the body of the first message in a conversation, useful for the bot to introduce itself or provide other initial context.
- **MessageFallback**: This will be printed if, for some reason, a regular Quixote message could not be generated. This might happen if all valid expansion paths contain contradictory conditions, e.g., a signature block with two options for first and last name in the case where neither has been defined. For a testing environment, you might want the Fallback message to be an explicit alert like "(Could not produce a response.)", whereas in a production environment, you might want the message to stay in character, like "I'm sorry, could you try rephrasing that?"
- **DefaultTransition**: This text will be printed between the FinishResponse message of one quest and the Assign message of the next. You can define more specific transitions for individual quests (see "transition" below).

Syntax:

**Symbol** **"***unique identifier with spaces***"** **"***quoted text(s)***"***...*

**Symbol** *UniqueIdentifierNoSpaces* **"***quoted text(s)***"***...*

Example:

*Symbol MessageGreeting "Hello,\n\n" "Hey there--" "Greetings,\n\n*
*Symbol "my custom symbol" "this is the one possible expansion for my custom symbol"*

### B.2.8 Persona

Defines behavior for when this bot should respond to messages. The first SchedulePersona defined will be the default. You can switch between them with an effect like *then {session.schedule_persona = "CaffeinatedStudent"}*

- **time range** is two clock times separated by **and**. Clock times can be written in most common formats: 10am, 3:45 p.m., 14:00, etc.
- **duration** is a length of time like *30 minutes* or *6 hours 10 minutes*. You can specify days, hours, or minutes in any combination. You an abbreviate these as d, h, m, like *6d 14h 30m*.
- **duration range** is two durations separated by **to**, like *10 minutes to 2 hours 45 minutes*.
- The bot will respond to messages received within the "available between" hours, after a random time within "duration range." If the "follows up after" time passes without receiving a response, the Followup move will be triggered.
    - Note that since times are randomly permuted for more believable behavior, the bot might sometimes reply a little bit after its cut-off point.
- The bot will attempt to follow up the number of times in provided in the "at most"… "time(s)" clause before giving up. If a message is responded to, this counter is reset. If 0, the bot will only reply to messages directly and never attempt to send a follow-up later. Each followup will be spaced out by the "follows up after" duration.
- If no schedule persona is defined, the default behavior will be to respond between 9am and 5pm after a couple of hours, and followup after 48 hours, one time.

Syntax:

**<u>Persona</u>** *PersonaName* [**Available between** *time range*] [**UTC** timezone] [**Response after** *duration range*] [**Follows up after** *duration*] [, **at most** *integer* **time(s)**]

Example:

*SchedulePersona CaffeinatedStudent*
*Available between 10:45am and 11:59pm UTC -7*
*Responds after 4 minutes to 22 minutes*
*Follows up after 4 hours, at most 1 time*

### B.2.9 Goal

Define a goal name that can be assigned within a quest.

Syntax:

**<u>Goal</u>** **"***Goal Name***"**

## B.3 Quest Blocks

A Quest represents a single "ask" from the bot, and perhaps some number of back-and-forth interactions discussing that request, ending with the interlocutor ultimately either fulfilling the request (i.e., by supplying some requested info) or failing to. When we move on to a different topic, that means transitioning to a new quest (even if a series of quests are linked together or conceptually related).

A Quest block begins with the word Quest, followed by the quest name in quotes. A second optional quoted string immediately following the quest name is a comment purely for the benefit of the author. A quest must define the move *Assign and generally defines other moves too. A project must have at least one quest to be valid, and a project is recommended to additionally have at least one *repeatable* quest as a fallback so it never runs out of content.

*Quest "Email Client Trouble"*

*"""Pretend we're having trouble with our email client."""*

** Assign "Your message seems to be corrupted somehow..."*

Quest parameters can appear in any order. As with other reserved words, they are case-insensitive.

- **Easy**, **Medium**, or **Hard**:
  - Indicate the difficulty level of the quest based on trust: Hard quests will only be assigned when trust is quite high, easy quests only when trust is fairly low. We will always pick quests from the most difficult category available.
  - If no difficulty is set, the quest will be available at any time regardless of trust.
- **Repeatable**
  - Indicates this quest can be repeated (default false). This is equivalent to setting all the quest's `* Assign` moves to be repeatable.
- **Priority** *Low, Normal [default], or High*
  - Indicates that all else being equal, a quest should be more or less likely to match. The weight is adjusted so any "High" quests will match before any "Normal" quests, and any "Low" quests will only match after every nonrepeatable "Normal" quest has been assigned. Useful for creating repeatable fallbacks in a difficulty category (e.g., all the regular quests are Normal, and the fallback quests are Low.)
  - Note: this does not overrule any other class consideration, so a Low priority Bait quest will still take precedence over a High priority non-Bait quest.
- **Captures**: optional list of 1 or more flags captured. Flags should be single words separated by spaces.
  - `captures ACCOUNT_NUMBER BANK_NAME`
  - Note that this does nothing by itself: you'll have to set up the flag capturing NLU by hand within information_extraction.py. It's just helpful metadata.

    Flags should sync up **transition** *SymbolName*
  - An expansion of one of the texts in the given symbol will be used to transition between a FinishResponse in a previous quest and the Assign/Reassign move in this quest. If no transition is defined for a quest, a symbol named DefaultTransition is used; if none is defined in your source file, one will be created upon export.
- **achieves "***DefinedGoal***"**...

    - List one or more defined goals that this quest achieves.
- **Test [optional number]**
    - Makes this quest come up first, regardless of trust.
    - If the release flag is set, Cervantes will halt with an error if any quests have a Test parameter.
    - If the optional number is given, it sets a priority, starting with lowest (i.e., Test 1 will be triggered before Test 2). This is useful for demos expecting a chain of quests in a certain order.
    - If multiple quests have a Test param without a number, they will be prioritized by the order they appear in the source file.
- **Disable**

Turns off a particular quest (equivalent to commenting it out).

**B.4 Quest Moves**

***** [*MoveType*] [condition(s) or ConditionList(s)...] [**repeatable**] [**:**] [**transition** *TransitionName*] **"***quoted text(s)***"**... [effect(s)...]

Defines the response for any specific move while this quest is running. The quoted text will be the response to the move. If you have multiple quoted texts, they will be selected from at random. You can use Productionist [[expansions]] in quoted texts: any that don't exist in your template project file will be created as starred nonterminals which can be filled in using Expressionist.

- Each move in a quest can only be triggered once per quest, unless "repeatable" is set. If the user response matches an already seen move for this quest, the system will fall back to *Refuse/*Giveup (if there are no Finish conditions that match) or a *Finish with no conditions.
- An *Assign or *Reassign move can specify a defined TransitionType to control what text will be printed between the FinishResponse of the previous quest and this move.
    - *Quest "URL Followup"*<br>** Assign when {temp.url exists} transition Unfortunately*<br>    *"that link still doesn't work."*
- You can also define multiple move names in a row if the response should be the same. When doing this, any conditions given will apply to all the move names.
    - **Yes *No *Refuse "I guess the answer doesn't really matter."*
- You can omit the move name to create a custom response. This is useful for creating quick one-off responses that don't need whole reusable move type definitions.
    - ** when {temp.surface.exclamation_mark} "There's no need to get upset!"*

- * *"I'm not sure I understand."*
- Just an asterisk with no condition will be a fallback, used if anything matches other than the defined quest moves. (Note that the default behavior for not understanding is to fall back to Giveup or Reassign based on trust, but this is useful if you want a wildcard response that doesn't end the quest.)
  - * *"I'm afraid I don't understand."*

## B.5 Conditions and Effects

Each condition or effect can appear alone or in a group. The general format is a type identifier followed by an expression in curly braces. You can reference a ConditionList by writing its identifier in curly braces. An inline list of conditions or effects only needs the type identifier before the first item in the list.

## B.6 Condition Types

- **when {***Productionist runtime expression***}**
  - These should be in the standard Productionist runtime expression format.
  - *when {flag.street does not exist}*
  - *when {x = True} {y = True} {z = True}*
- **when {***ConditionList Name***}**
  - Equivalent to pasting all the conditions defined in the given ConditionList here.
- **before {***Quest Name***}**
  - This quest can only be assigned if the given quest has never been assigned.
- **after {***Quest Name***}**
  - This quest can only be assigned after the given quest has been assigned, but not immediately after. This is useful to create a quest that's reopening a previous subject.
- **next after {***Quest Name***}**
  - This quest can only be assigned immediately after the given quest, useful for chaining steps in a multi-part quest. Note that this condition will not pass if the user explicitly Refused to do the parent quest.
- **maybe after {***Quest Name***}**
  - This quest has a 50% chance of being assigned immediately after the given quest, useful for nondeterministic direct follow-ups to a quest like nitpicking with the response some of the time.

If multiple responses of the same type match, the earliest defined in the code will be the one selected. This means you should put more specific matches first. If you want a more specific match to show up only some of the time, you might also add a `temp.chance` condition (see below).

- *# Ensure a state-specific response matches before a country-specific response.*
  ** Finish when {flag.state == 'FL'} "…"*
  ** Finish when {country == 'US'} "…"*

### B.7 Effects

Effects make a change in the system state. When effects are part of a move response definition, they will run after that move has been selected and its text generated.

- **then {***Productionist runtime expression***}**
  - ** Assign: "Let's play a game." then {status.game = 'begun'}*
- **then {continue quest}**
  - If this move type is a FinishResponse that would normally end the quest, instead keep it running. The word "quest" can be omitted.
  - **Refuse: "I really won't take no for an answer." then {continue}*
- **then {end quest}**
  - If this move type is a Response that would normally continue the quest, instead end it. The word "quest" can be omitted.
  - **Question: "I said no questions! Moving on…" then {end quest}*

## APPENDIX C - Symbol Name Conventions

This is a schema for naming nonterminal symbols within an Expressionist template to keep them organized and consistent. The schema works like a **contract** so the author can have confidence that using a symbol in a particular place will expand to text with predictable qualities.

- **Symbols should use internal spaces and punctuation when appropriate**, ideally looking like a normal English phrase that might be one of the expansions. If the usage is ambiguous they should try to clearly indicate when and how the symbol should be used.
  - Bad: [[Id]] (does this mean ID or "I'd"? If the latter, I would or I could?)
  - Better: [[I would]]
  - Bad: [[ht_name]] (what's "ht" short for? Are there trailing/leading words?)
  - Better: [[the random hotel name]] -> "the Marriott", "the Quality Inn"
- **Begin with a capital letter** if the symbol expands to text that begins a sentence.
  - [[Greetings]] -> "Hello"
- **Begin with a lowercase letter** if the symbol text can be used in the middle of a sentence.
  - [[thanks]] -> "thank you"
  - Note that you can use {+} right before a letter to change the casing, which is useful if you want to start a sentence with a random phrase.)
    - {+}[[thanks]]!
    - In general you should go from lower to upper case, because the opposite can produce weirdnesses around "I", like: {-}[[I could]] -> "i could"
- **End with a comma** if the symbol ends a clause and a new phrase can begin next.
  - [[Anyway,]] -> "By the way", "In other news,", "Hey--"
- **End with a period** if the symbol concludes a sentence.
  - [[Greetings.]] -> "Hey there!"
  - (Note that these rules all stack, as in the example above.)
- **Begin with an underscore** if the symbol is designed for internal use by another symbol and shouldn't be directly invoked in dialogue (unless the author is doing so intentionally).
  - [[_greeting when scared]]
- **Begin with "random"** if the symbol selects between a set of named alternatives.

    - [[random company]] -> “Chevron”
- **Begin with “maybe”** if the symbol sometimes prints text and sometimes prints nothing.
    - [[maybe smiley]] -> “=)”, “”
- **Begin or end with a dash** if the symbol’s expansions should have a leading/trailing space.
    - Unfortunately it didn’t work[[-maybe apology]].
- **Begin with “your” or “my”** for expansions related to the listener or speaker (bot):
    - [[your first name]] -> “{attacker.name.first},” “buddy”

## APPENDIX D - State Schema

The global state uses a hierarchy of keys:

- **global.***
  - Anything under this header will be available in all conversations across the running system, not just this one: useful for cross-bot coordination.
  - For the BBN SIENNA eval, have defined two variables, one of which is expected to exist and be set to True at startup (the other should not exist)
    - global.domain.vt
    - global.domain.sri
- **temp.***
  - Anything stored here will be erased at the end of the current round of dialogue generation.
  - **temp.nlu_acts.*** Results of natural language understanding operations
  - **temp.nlu_calculated.*** Results of calculated NLU acts based on combinations of lower-level signals.
  - **temp.phrases.***
    - **temp.phrases.topics** A list of topics that were referenced in this message.
  - **temp.surface.*** Documenting surface-level features of NLU, like "does it have a question mark"
  - **temp.chance.p10, .p25, .p50, .p75, .p90** If these are true, a random chance of that percentage value is true on this turn.
  - **temp.has_date** A date was just mentioned (stored in last.date.*)
- **last.***
  - The most recent item of each type mentioned. Note that this may not mean this was mentioned in the most recent message: to look for a signal like that, use a temp.* variable which is cleared at the end of each turn.
  - **last.browser**
  - **last.email**
  - **last.linkedin_username**
  - **last.url**
  - **last.date.year, .month, .day, .day_of_week, .date, delta.days, .time**
  - **last.address.full, .full_street, .street_number, .street_name, .street_type, .street_direction, .floor, .apt_number, .city, .state, .province, .country, .postal_code**

- **last.dollar_amount**
- **bots.***
    - Functions to get information from or invoke a language generation bot.
- **flag.***
    - Info about flags captured from the interlocutor.
- **last_attachment.***
    - Info about the most recent attachment in an interlocutor message.
- **last_link.***
    - Info about the most recent link in an interlocutor message
- **request.***
    - SIENNA detected the human has asked for something.
    - **request.connect_on_linkedin**
    - **request.pivot_to_sms**
- **date.***
    - Info about the date of the most recent message.
    - **date.date** = “10/06/20”
    - **date.day** = 1-7
    - **date.day_of_week**
    - **date.month**
    - **date.year**
- **quests.***
    - Info about the state of quests.
    - **quests.current.*** Info about the current quest; cleared when a new one is assigned.
        - **quests.current.ID** id of current quest
        - **quests.current.counter** number of turns since this quest was assigned
        - **quests.current.moves_played** array with the names of all quest-specific moves played since this quest began.
        - **quests.current.last_move** the most recent move played.
        - **quests.current.soliciting** a string with a type of info we have asked for: valid options are “image”, “document”, “url”. Useful for distinguishing solicited vs unsolicited information.
    - **quests.previous** Array of all previously assigned quests.
- **session.***
    - Info about the current session.
    - **session.earliest_time, session.latest_time**

- **session.schedule_persona**

- **debug.***
  - Info for demo interface.

There are also a bunch of Productionist-only state items that aren't part of the actual state object but can be used in dialogue as if they are. These are created in conversationalist.py _get_temporary_production_state_overrides.

- **me.***
  - Info about the bot's persona.
  - **me.name.first, me.name.last, me.name.full**
  - It's safer not to use these variables directly, but (if you're using the default discourse_library) to use the expansions [[my first name]], [[my last name]], and [[my full name]]. These will print a sensible default if a value happens to be unset.
- **you.***
  - Info about the interlocutor.
  - **you.name.first, you.name.last**
- **last.***
  - Info about the most recently sent interlocutor message.
  - **last.media_type**
  - **last.when**
    - **.year, .month, .day, .day_name, .weekday, .weekend, .morning, .afternoon, .evening, .overnight, .hour**
- **messages_sent.me, messages_sent.them**
- **rng.coinflip**
- **control.***
  - Info about timing and message format.

# APPENDIX E - QUEST LIBRARY

| Name | Quest Type | VT-Domain Specific | Difficulty | Repeatable | Follow-up to | Flag? | NLU Needs? (see NLU Wishlist) Green exist already. | Status (Concept, Stub, | Target asks... | Time wasting potential: attacker believes they must... | Actual validation criteria |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **Email Client Trouble** | Delaying | | Easy | | | | | Complete | for message to be resent | check their email program settings and resend their message | none |
| **Advanced Email Client Trouble** | Mission | | Medium | | Email Client Trouble | | | Complete | for attacker to switch to a different device or email client | investigate possible sources of error | none |
| **Fallback Email Client Trouble** | Fallback | | Easy | Y | Email Client Trouble | | | Complete | for message to be resent | diagnose problem and/or resend message | none |
| **Deja Vu** | Delaying | | Easy | | | | | Complete | if someone else from attacker's org has already contacted them about this | prove they are not just another scammer | none |
| **Mailing Address** | Flag Capture | | Medium | | | STREET, CITY, ZIP, POSTAL_CODE, COUNTRY | Address detection | Complete | for mailing address of attacker or attacker's org | share their real address or fake a plausible one | address found |
| **Address Chit-Chat** | Delaying | | n/a | | Mailing Address | | | Complete | a follow-up question about the attacker's address | respond to the chit-chat plausibly | none |
| **Capture LinkedIn** | Flag Capture | | Medium | | | linkedin_username | Named entity recognition | Blocked [NLU] | for attacker's LinkedIn account | share this info or fake something | detecting something that looks like an account name |
| **Capture Age** | Flag Capture | | Easy | | | age | number in plausible range | Complete | for attacker's age | share this info | a number between 12 and 100 |
| **Birthday** | Flag Capture | | Medium | | | AGE, DOB | Date detection (with or without year) | Partial [NLU] | for the attacker's date of birth. | share this info or sensibly refuse | date and/or year found |
| **Nemesis** | Flag Capture | | Easy | | | various | | Complete | for the attacker to click a NEMESIS link | if they click the link, info about them is captured | none |
| **General Area** | Flag Capture | | Easy | | | CITIZENSHIP, COUNTRY, STATE | detecting country, state, or citizenship words (i.e. "Italian", "I live in Madrid", "I'm a Texan") | Blocked [NLU] | where the attacker is living or from | share this info or sensibly refuse | recognized response |
| **Quibble with Quote** | Fallback | | n/a | Y | | | | Complete | for a reponse to a quibblebot-generated comment on a random phrase from the last message | spend several moves replying to quibblebot | none (transitions back to Quixote after 3 moves) |
| **Connect on Social Media** | Mission | | Medium | | | | Compliance ("I've done what you requested") or planned compliance ("Sure, I'll get right on that.") | Complete | to connect with the attacker on an obscure social media network | create an account there to continue the conversation | signs of compliance |
| **Social Media Troubles** | Mission | | n/a | | Connect on Social Media | | | Complete | for troubleshooting with connecting with the attacker's new social account | diagnose why they can't connect with the target | none |
| **Secure App** | Mission | | Hard | | | | | Complete | to connect via an obscure secure messaging platform | sign up for an account | none |
| **Secure App Connection** | Mission | | n/a | | Secure App | | | Complete | to accept their invite on the secure platform | try to make a connection with the target on the platform | none |
| **Secure App Troubleshooting** | Mission | | n/a | | Secure App Connection | | | Complete | to continue troubleshooting their inability to connect on the secure platform | keep trying to make the connection happen | none |
| **Qualifications** | Delaying | | Easy | | | | | Complete | why the attacker contacted them in particular | explain why they picked the target | none |
| **Business URL** | Flag Capture | | Easy | | | DOMAIN | URL detection | Complete | for a URL for the attacker's org | share a URL or create a plausible fake website | message contains a url |
| **Complain about URL** | Delaying | | n/a | | Business URL | | | Complete | for the attacker to resend the URL because it didn't come through | resend the URL | message contains a url |
| **Complain about Document** | Mission | | Medium | | | | attachment detection | Complete | for a requested document to be resent in a different format | redo the document to the arbitrary parameters specified | message contains a document |
| **Complain about Image** | Mission | | Medium | | | | attachment detection | Complete | for a requested image to be resent in a different format | redo the image to the arbitrary parameters specified | message contains an image attachment |
| **Capture Age** | Flag Capture | | Easy | | | AGE, DOB | Number detection (Cervantes code can restrict to plausible age ranges) | Draft | for the attacker's age | share or fake this info | message contains a plausible number |
| **Get Time Zone** | Flag Capture | | Easy | | | LOCATION_TIME_ZONI | Time Zone references (i.e. GMT +1, EST, Pacific Time, etc) | Planned | what time zone the attacker is in | share or fake this info | message contains a time zone |
| **Get Fluency** | Flag Capture | | Medium | | | LANGUAGES | names of languages (Spanish, French, etc.) | Complete | what languages the attacker speaks | share this info accurately or risk being called out | message contains one or more language names |
| **Charitable Donation** | Mission | | Hard | | | | | Complete | for the attacker to make a small donation to a charity as a sign of good faith, and send a copy of the donation receipt | make the donation and send a receipt as requested, or explain why they're not willing to | response contains an attachment |

| Name | Quest Type | VT-Domain Specific | Difficulty | Repeatable | Follow-up to | Flag? | NLU Needs? (see NLU Wishlist) Green exist already. | Status (Concept, Stub, | Target asks... | Time wasting potential: attacker believes they must... | Actual validation criteria |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **Ask for Photo** | Mission | | Hard | | | (potentially PHOTO_URL) | | Complete | for a photo of the attacker holding a sign with their name | take this photo or fake one | response contains an image attachment |
| **Unsolicited** | Delaying | | n/a | Y | | | attachment detection | Complete | for an explanation of an unsolicited attachment | justify the attachment | 3 turns have passed |
| **Unsecure Domain** | Bait (URL) | | Hard | | | | URL detection | Complete | If they could provide the info at a domain not flagged as suspicious | provide a new malicious link, or giveup | new link is at a different domain |
| **Browser for Link** | Bait (URL) | | n/a | | | | URL detection | Complete | What browser they should view the link in? | explain | mention of a web browser |
| **Browser Followup** | Delaying | | n/a | | Browser for Link | | | Complete | If there's a different browser they could use because they don't have the one provided. | explain | none |
| **Unlickable Link** | Bait (URL) | | n/a | | | | URL detection | Complete | If the link could be resent because it's not clickable. | explain how links work or resend as HTML link | link is resent |
| **Malformed Link** | Bait (URL) | | n/a | | | | URL detection | Complete | If the link could be resent because it came through corrupted. | | |
| **Empty Page** | Bait (URL) | | n/a | | | | URL detection | Complete | what they expected to see at the link page because it came through blank | | |
| **Blacksburg Establish Location** | Bait (Blacksburg) | Y | n/a | | | (potentially city) | mention of Blacksburg | Complete | for the attacker to say if they're from Blacksburg or not | | Yes/No |
| **Blacksburg Local Chitchat** | Delaying | Y | n/a | Y | Blacksburg Establish | | | Complete | for the attacker's opinion about a local Blacksburg issue | come up with a plausible response | none |
| **VA Tech Chitchat** | Bait (VA Tech) | Y | n/a | Y | | | mention of VA Tech | Complete | a followup question about Virginia Tech | come up with a plausible response | none |
| **Info Extraction Class Chitchat** | Bait (class) | Y | n/a | Y | | | mention of the Information Extraction class | Complete | a followup question about a reference to the class | come up with a plausible response | none |
| **Covid Chitchat** | Bait (covid) | Y | n/a | Y | | | mention of coronavirus | Complete | a followup question about covid | come up with a plausible response | none |
| **Scheduling Hell** | Bait (meeting | | n/a | Y | | | mention of words like "schedule", "meeting", etc | Complete | to try to schedule a meeting time | play scheduling tag | 3 turns pass |
| **Qualifications** | Delaying | | Easy | | | | | Complete | why they were contacted in particular | rationalize why they're contacting the target | none |
| **Personal Question** | Bait (personal question) | | n/a | | | | Personal question detector (Followist) | Planned | expresse offense at being asked a personal question, demands an apology | placate the target or justify why they want the information | none |
| **Capture Appearance** | Flag Capture | | Medium | | | EYE, HAIR | regex for common eye/hair colors | Planned | the attacker to supply demographic information in the form of a "Your X Name Is Y + Z" game | | |
| **Capture Occupation** | Flag Capture | | Easy | | | OCCUPATION, COMPANY, TITLE | named entity recognition | Planned | what company the attacker works for and what their position is there | reveal or invent the info | named entities that could be a company name or job title |
| **Captured LinkedIn** | Flag Capture | | Easy | | | LINKEDIN_USERNAME | named entity recognition | Planned | to connect on LinkedIn | reveal their LinkedIn name or spend time making a fake page. | named entity that could be a LinkedIn profile name |
| **Missing Link** | Bait (URL) | | n/a | | | | URL detection | Complete | for link to be resent, since it was stripped from the message for some reason | figure out the reason the link they're sending isn't coming through | none |
| **Link Captcha** | Bait (URL) | | n/a | | | | URL detection | Complete | for the attacker to say something verifying they're a real person | respond with something human-y | none |
| **Personal Question** | Bait (TMI) | | n/a | | | | Detect an overly personal question | Complete | for the attacker to justify why they're asking a personal question | explain themselves | none |
| **Another Personal Question** | Bait (TMI) | | n/a | Y | Personal Question | | Detect an overly personal question | Complete | why the attacker keeps asking inappropriate questions | explain themselves | none |
| **Respond to Legal Action** | Bait (Initial) | | n/a | | | | Detect a legal trouble/attorney scam | Complete | the attacker to send more info | send additional info | none |
| **Respond to Financial Offer** | Bait (Initial) | | n/a | | | | Detect a financial offer/banking scam | Complete | the attacker to send more info | send additional info | none |
| **Respond to Recruiting** | Bait (Initial) | | n/a | | | | Detect a job offer / recruitment scam | Complete | the attacker to send more info | send additional info | none |
| **Respond to Alumni Scam** | Bait (Initial) | Y | n/a | | | | Detect a commencement / alumni scam | Complete | the attacker to send more info | send additional info | none |
| **Respond to VT Communication** | Bait (Initial) | Y | n/a | | | | Detect an impersonating Virginia Tech administrator scam | Complete | the attacker to send more info | send additional info | none |