nXcomms: Advancing Communications with Virtual Patients

Jerry Heneghan, Brandon Conover,
Christine Heneghan, Chris Wall, Jason Cisarano,
Wesley Pretsch
BioMojo, LLC, Cary, NC
{jerry, brandon, christine, cwall, jason, wes}
@biomojo.com

William N. Vasios, III
WV3 LLC, Holly Springs, NC
wnviii@gmail.com

Gene Hobbs, Robert Hubal
University of North Carolina
Chapel Hill, NC
{gene, hubal}@unc.edu

ABSTRACT

The intent of this project was to use natural language processing and AI features for medical simulation training to augment the reality of medical manikins as well as screen-based patient simulators. The work involved defining, developing, testing, and demonstrating an intelligent patient simulation software architecture to provide realistic medical training experiences. The solution involves two parallel components—first the integration of language and intelligence models for users and observer/controllers (O/Cs), and second the design for current and future integration of smart interaction to take additional advantage of technological capabilities.

For the first component, software enables users to naturally ask questions and receive responses from the simulator (whether hands-on or screen-based), and it allows the simulator to carry out commands issued by the O/C. The user and O/C are able to carry out simulation dialog without having to speak in pre-defined manner. The software was designed to integrate seamlessly with both manikins and screen-based patient simulators widely in use across the DOD. It also integrates with existing intelligence modules such as physiology engines, behavior models, and adaptive testing. For the second component, the interaction allows users to engage with the simulated patient and an O/C to control the simulation system. The methods facilitate multimodal dialog and feedback between the user and system, working with the user to understand, clarify, and put into effect the user's intent.

This paper describes the development and testing of the architecture.

ABOUT THE AUTHORS

Jerry Heneghan is Chief Design Officer at BioMojo, a medical simulation innovator, pioneer in computational biology and a creator of virtual physiological humans for education, training and research.

Brandon Conover has an extensive background in theoretical and applied electromagnetics and leads multiple Warfighter-facing technology development efforts.

Christine Heneghan is co-founder of BioMojo, serves as its CEO, and has extensive experience in interactive entertainment, serious games, and the performing arts.

Chris Wall has 25 years' software engineering experience in the game and simulation industries, with recent experience in developing AI driven natural language processing applications.

Jason Cisarano researches machine learning solutions for natural language and computer vision problems and builds 3D virtual worlds for games, simulations, and AR devices.

Wesley Pretch is studying Computer Science at North Carolina State University. He is pursuing software development with focuses on AI and Machine Learning.

Bill Vasios is a lifelong learner with a keen interest in sharing 34+ years of Special Operations experience and 26+ years as a medic and Physician Assistant across multiple specialties.

Gene Hobbs is a Certified Healthcare Simulation Educator with over 20 years of experience in training and development. He is also the business manager for the UNC Department of Neurosurgery.

Rob Hubal has long been interested in simulation and natural language tools used for training and education.

nXcomms: Advancing Communications with Virtual Patients

Jerry Heneghan, Brandon Conover,
Christine Heneghan, Chris Wall, Jason Cisarano,
Wesley Pretsch
BioMojo, LLC, Cary, NC
{jerry, brandon, christine, cwall, jason, wes}
@biomojo.com

William N. Vasios, III
WV3 LLC, Holly Springs, NC
wnviii@gmail.com

Gene Hobbs, Robert Hubal
University of North Carolina
Chapel Hill, NC
{gene, hubal}@unc.edu

INTRODUCTION

A team of researchers and military and civilian medical subject-matter experts (SMEs) created software called nXcomms that adds natural language (NL) processing and artificial intelligence (AI) features to simulated patients—both medical manikins and screen-based virtual patient simulators. The software is intended to provide enhanced, realistic medical training experiences, including capabilities for learners to ask questions and receive responses from the simulated patients in NL and for observer/controllers (O/Cs) to dynamically control the simulation, including changing the underlying physiology. AI functions allow this software to learn from iterative training events. To date, this team has defined, developed, internally tested, and demonstrated the software, using a parallel path of two main components—first the integration of language and intelligence models for users and for O/Cs, and second the design for future integration of smart interaction to take additional advantage of technological capabilities.

For the first component, software was developed to allow users to naturally ask questions and receive responses from the simulator (whether hands-on or screen-based), allowing the simulator to carry out commands issued by the O/C. The user and the O/C are able to conduct dialog with the simulator without having to speak in certain pre-defined sentences. The software has been designed to integrate seamlessly with both manikins and screen-based patient simulators widely in use across the DOD. It also integrates with intelligence modules (e.g., behavior models; physiology engines) that team members have worked with (Morbini et al., 2012; Sottilare et al., 2017) and even helped develop (Bray et al., 2019; Kizakevich et al., 2006).

For the second component, the concept of smart interaction that was originated by Taylor et al. (2012) was intended to improve how a user may engage with the simulated patient or how an O/C can control the simulation system. The methods facilitate a kind of dialog between the user and system, working with the user to understand the user's intent, including asking any clarifying questions. nXcomms was designed to allow for the future incorporation of multiple input modalities, including speech, gesture, and sketch as natural ways for a user to communicate with intelligent systems, and to support multiple modes of feedback, including graphics, video, and speech. The current design and future plan are the integration of smart interaction within DOD medical simulation training.

A proof-of-concept demonstrated that NL software will work in a DOD-simulated patient training environment to answer questions posed by users and carry out commands given by the O/C. The design used high-level specifications such that it would work on multiple platforms in use at DOD simulation centers. This paper reports on the system's feasibility, usability, reliability, and capability.

TECHNICAL TASKS

The work described in this paper was divided into two technical tasks, (1) smart interaction and integration and (2) field analysis.

Task 1—Smart Interaction

NL Processing

The first of this task's objectives was to implement NL capability for typical DOD medical simulation training environments, and determine questions and commands that need to be captured by the language processor and subsequently map spoken input to an underlying intention and produce responsive output. Desired outcomes were to accurately understand questions and commands in different forms, and respond appropriately, as in natural conversations, via spoken words, speech recognition, semantic mapping AI, and speech synthesis response.

Programmatic and software requirements leveraged much prior language processing work (Guinn & Hubal, 2003; Hubal & Heneghan, 2017). As in past projects, spoken input is mapped to an underlying 'intent' that produces responsive output. Simulated personalities are tracked through the branching dialog and also through trust tracking. The patient linguistic responses are defined in Ink (github.com/inkle/ink; an open source software tool) dialogs, and trust represents the cumulative effect of interactions throughout the encounter. For example, if the user has a high trust rating with the virtual patient, then the patient may lean forward and respond positively to the question asked. If the user has a low trust rating, the virtual patient may lean back with her arms crossed in front of her and reply tersely. Patient interaction is conveyed to the user through dialog, facial expressions, and body language. Combinations of postures, facial expressions, and gestures are used to convey various emotions.

As is typical of NL applications, the design began by creating a workflow for developing branching dialog scripts and leveraging SMEs for the content. The workflow model divided the creation of scripts from their implementation in Ink, which allowed for the incorporation of SME input to specify patients' personalities and histories along with the speaker's primary areas of focus. The script writer used this information to construct interactive conversation scripts in an implementation-independent form and the scripts could then be implemented in Ink by an independent developer.

There are three parts to the system's NL processing: speech-to-text, language understanding, and patient response.

- Speech-to-text. The advanced deep-learning neural network algorithms developed by Google are used to convert microphone audio input-to-text. Google's service is not unique and any of several services (e.g., from Amazon, Apple, or IBM) could be used. The audio is streamed to Google's servers and immediately returns text as it is recognized. The service also handles noisy audio from many environments without requiring additional noise cancellation. The system is modular so that, even though it leverages this industry-standard process, it could easily support other solutions, including an offline option.
- Language understanding. Upon receiving the text result, the NL processor intelligently matches the text against a list of previously generated intents. The dialog scripts in Ink are used to constrain the expected inputs at any given time and thus to maximize hits. In general, there would be only a handful of reasonable inputs—intents—at a given time, and thus a limited number of expected directions during any natural dialog. To conduct the match of actual input to intents, a list of phrases is needed that signify each intent. In this work, phrases were generated using several procedures: (i) taking from transcripts of real conversations, such as from structured clinical examinations and from SME discussions; (ii) use of Mechanical Turk whereby people were asked to provide alternative phrasing for intents; and (iii) NL training with a team of testers to seed a database. Intents are matched by comparing each spoken word against each word in the intent. A stemming algorithm is used for linguistic normalization, and filter stop words (e.g., 'and', 'the'). Longer words are given more weight than shorter words; bonus weight is awarded if the spoken phrase is contained in the intent. Unless it finds an exact match, the system looks at all intents to search for the best match. (Three string distance algorithms, Sørensen–Dice, Levenshtein, and NGram, were tested; none improved on the existing approach.) If no option is found, the algorithm highlights any topic hints and causes the virtual patient to say a clarification phrase, such as "Could you rephrase that?" The system automatically adds the intent if not already in the intent dictionary.

  New intent matching features were developed to improve support for the "always listening" environment required when talking to a wounded soldier and to reduce false positives. These include adding support for 'must include' words for individual examples; extended NL to improve resolution of very similar intents; and detection of questions versus statements that do not require a patient response. Finally, the system is aware of medical procedures, physiology terminology and common medical parlance. That is, it provides (i) support for various volumes per time period ("Start hemorrhage at 50 mL per minute."); (ii) support for various masses per volume period ("Inject 5 mL of morphine subcutaneous at a concentration of 1 mg per mL."); and (iii) hints as to what the user is likely to talk about to include mass and volume units, reducing the chance of the algorithm misinterpreting what was said.

  Branching conversation was improved by several adjustments: (i) automatic selection of a specific response if the learner does not say anything for a set period (included option for this automatic response to not be visible to the user); (ii) support for falling through conversation choices to the next set of choices (e.g., if a choice is "Discuss medications" and "Allow Fall Through" is set, and the user speaks, "Tell me about any prescriptions you are taking," the system would first match against the "Discuss medications" option and proceed to load the next choices; if one of the next choices was indeed something like "Tell me about your prescriptions" then that would be triggered too); this system was also used to support the user asking multiple questions in the same utterance; and (iii) prioritizing and giving extra weight to defined keywords per intent.

- Patient response. The Amazon Polly service is used to convert the patient's text response to an audio file for playback. Amazon Polly (again, among other like services) uses deep learning technologies to synthesize speech that sounds like human voice. For the screen-based virtual patient, the voice playback is synchronized to the virtual patient's lips movement using actual phonemes of the spoken text (a set of lip shape animations model actual lip shapes of spoken English) to create a more immersive user experience. The system supports multiple voices for different purposes (e.g., the manikin, virtual patient, and O/C feedback all have distinct voices). In addition, the service can add certain special effects like coughs and pauses. Audio files are cached at two levels for improved performance. First, they are cached on a server, decreasing latency for later users who interact with the same character. Second, they are cached locally on the user's machine, so if the user runs through the encounter a second time, there is no need to fetch the file from a remote server. In the occasional case of a failed response, a smart system helps nudge the user in the direction of a successful response. For example, if the microphone volume is low, the virtual patient asks the user to speak up. Meanwhile, if the response is unintelligible, the patient asks the user to rephrase their statement. In addition, software performance was profiled, including optimizing audio tracks for streaming, reducing memory footprint by 80%.

Supra-Linguistic Processing

The second of this task's objectives was to investigate meta-linguistic input (i.e., the speaker's means of communication, beyond the content), and important movement (i.e., the speaker's movement) during user-patient interactions and O/C-patient interactions. The initial step was observing standardized patient training and actual patient interactions at the University of North Carolina (UNC) School of Medicine and, from those observations, identifying meta-tagging needed to indicate actions that a virtual patient should make, such as specific gestures or expressions. Actions taken on the manikin, such as pulse rate or needle decompression, are captured and translated into informed responses. Embedding conformable sensor technology was explored to provide real time assessments, further enhancing the skill development supported.

An audio input system was also designed and implemented. It supports the user and the O/C talking to the system simultaneously, understands their spoken words and takes the appropriate action. The system involves: (i) accepting input from two separate microphones and funneling the input to the correct NL; (ii) optional playback of the user's speech to the O/C to support the user being a distance from the O/C; (iii) support for both the O/C and the user to hear the patient's response; (iv) support for only the O/C to hear responses from their NL physiology commands; (v) audio files returned from the text-to-speech module that are cached to the local hard drive to reduce network usage and improve latency in NL responses; (vi) O/C ability to hear what the user is saying to the patient as well as the patient's reply; (vii) O/C ability to use speech-to-text to record notes; (viii) O/C ability to control the physiology engine running on the manikin in real time using voice commands; (ix) O/C ability to control manikin systems directly if needed; (x) incorporation of microphone-recording decibel-level controls as well as separate volume controls for O/C and user speakers; and (xi) optionally saving user voice-audio recordings to disk or a server for NL training. Additionally, Amazon Polly support was updated to include Speech Synthesis Markup Language (SSML). Developers can use Amazon Polly to generate speech from either plain text or from documents marked SSML. Using SSML-enhanced text gives additional control over how Amazon Polly generates speech from the text provided. For example, it is possible to include a long pause within the text or change the speech rate or pitch. Other options include emphasizing specific words or phrases; using phonetic pronunciation; and including breathing sounds and whispers. Amazon Polly provides these types of control with a subset of the SSML markup tags that are defined by the World Wide Web Consortium (W3C, 2010).

The system is built to work well with virtually any manikin/simulator environment. It is possible even to select the manikin to connect to; each manikin has a unique identifier based on the computer controlling it and its IP address. nXcomms accommodates Representational State Transfer (REST) application programming interface (API) access, as well as response from Unity applications, and has built-in server-side forwarding to support REST calls. REST style interfaces are an enabling feature for web services, which in turn make data access and control information from elsewhere on the network available to web applications and mobile apps. RESTful APIs have a number of defining characteristics. Pertinent are the stateless interface, such that requests provide all the information required to perform the requested service; layerability, so that the clients do not know if they are directly connected to the service or if there are intermediate layers that provide features such as scalability and security; and uniform interface, which provides a simple and complete method for accessing and manipulating the service. Adding a RESTful API provides a standard mechanism for any web service or tool to access and manipulate medical simulation. Some of the features developed for the server RESTful API include tracking choices per user to allow for recreating any training session in detail and recording lines the user spoke as speech-to-text along with any matched and unmatched intents. These

tracking and recording data are used to (i) train the language processing AI; (ii) send data, in the unlikely event of an error in the application, to the server; (iii) send additional information such as scenario and encounter identifier; and (iv) cache to reduce data signal noise while reducing bandwidth.

Technical Components

- Physiology engine. The Pulse physiology platform (Bray et al., 2019) supports the design, development, and use of physiologic modeling for building medical training content. The goal for Pulse is to lower the barrier to entry for medical training and simulation developers by supplying an open-source physiology engine that produces reliable, accurate, validated physiologic responses. The architecture is specifically designed to reduce model development time and increase the usability of the engine in simulations by creating a modular, extensible ontology for simulating the human physiology. The Pulse platform, a fork of the BioGears physiology engine, is accessible via a public repository that has a number of users and outside contributors to foster a true open-source community. The Pulse physiology platform has also been integrated into a number of research, clinical, and commercial applications.

  Pulse is comprised of lumped-parameter models, which use circuit analogs (i.e., resistors and capacitors) to represent the behavior of a region of interest/system of the body. Incorporation of feedback is accomplished by modeling feedback mechanisms and applying the effects to the circuit components. The physiology engine also includes pharmacokinetic and pharmacodynamic models, which use the physiologic properties of the patient and the physiochemical properties of the drug in differential equations to represent drug diffusion and distribution in the body. Disease and treatment models are designed with differential equations representing the effects of disease and treatment and then applied to the lumped-parameter models to affect the overall calculations. Complete scenario evaluation is conducted by constructing and executing scenarios that represent disease and treatment actions. A matrix of injuries, disease, and/or treatments is constructed with timelines and expected physiologic outcomes. The physiologic model behavior is compared to the expected behavior to ensure trends, timeframes, and magnitude of responses are valid. Pulse can be incorporated into the framework of any medical tutorial or diagnostic support tool to project the condition of the patient as time progresses.

  The Pulse scenario capability sets up initial conditions for use cases. Vitals updates are received from Pulse and displayed in a user interface (UI) element and to any attached manikin. Data and commands are sent back to the physiology engine, when, for example, the user performs a needle decompression.

- Manikin layer. An abstraction layer was created to support multiple manikins. For this effort, the system was designed with the Laerdal SimMan® 3G, using the non-commercial LLEAP software development kit (SDK)/API ver 6.7 and subsequent updates. This SDK is in active development by Laerdal, and it gives access to all advanced features of any Laerdal manikin, including vital signs, session logs, and manikin sensors. The current application accesses the SDK via a RESTful API, but it can also be accessed via desktop applications in C# and C++.

- Interface layer. To support other physiology engines and manikin types, an interface layer converts Pulse physiology data to internal abstraction data structures and then converts internal abstraction data structures to the manikins' data structures. Nothing in the internal system is tied to the Pulse engine—there is always a layer of abstraction between the system and the external engine. The same is true for the manikin interface in that the application is not tightly coupled to the Laerdal manikin SDK.

Task 2—Integration and Field Analysis

This task's objectives were to integrate the system components into a simulation center and analyze results by addressing (i) assistance to the operator of a simulation who must manipulate the scenario; (ii) capture of questions and commands; (iii) integration of AI modules to support NL; and (iv) leveraging NL processing to overcome traditional input restrictions.

Integration of the components into a simulation center was shown in two parallel and modular paths, virtual patient and interactive medical manikin. The approach is modular; while a particular physiology engine (Pulse), specific manikin (Laerdal) and its related SDK, and a single language platform for translation were chosen, the platform itself can readily incorporate many different choices from each of these categories. Working with military and civilian medical SMEs, a scenario was crafted that meets current operational needs. The scenario was presented to the sponsor for demonstration utilizing both a virtual patient and a manikin. On both paths, the focus was equally on (i) making the casualty communicative and (ii) making it easy for the O/C/trainer to manipulate what is presented to users via voice to lessen cognitive and behavioral burden. For both the virtual and manikin approaches, the incorporation of a physiology engine provides additional realism and training value.

Scenario Development

There was much overlap with slight differences between the virtual patient and interactive medical manikin. To make the system function requires building scenarios. Scenarios require populating physiology and other AI components with situation-specific data. Scenarios also require the specification of criteria to assess 'success'—that is, the learning objectives in an educational setting. As examples: Did the user address all important topics to sufficient depth? Did the user avoid incorrect behaviors? Populating components defines what actions should occur under what conditions.

For the NL component, the scenario consists of a clinician interacting with a patient using branched conversations. Developing this dialog requires creating a script of the 'best case' interaction between the provider and patient or system and O/C; coding the script in Ink; generating alternative phrases for every conversational turn in the script; testing the dialog by bringing in SMEs and other providers; revising the dialog in the NL engine based on providers' input; and repeating the last three steps until a nearly complete conversation is done. The conversations and responses from the patient were carefully considered, should they elicit stress or emotional responses from the user. The responses from both the patient and user—as well as the dialog between the system and simulation O/C—are logged for subsequent analysis. A trust indicator shown during the interaction and a comprehensive after-action review linking user actions against the aforementioned success criteria are meant to allow users to understand the effects of their actions and O/Cs to better engage user performance.

The UI sections are: (i) buttons, through which the virtual patient has a display of the choices from the conversation tree that are available at that particular time. The user can click on one of these buttons or talk in a natural way on a topic related to that button. If the button states "Ask for purpose of visit" the user can say, for example, either "Tell me about the reason for your visit." or "What brings you in today?"; (ii) trust meter to dynamically show the extent of rapport that the learner has built to that point; (iii) microphone readiness indicator that turns amber when it is activated and green when it detects the user talking. It is red and disabled while the patient is responding; and (iv) chat log to display all of the questions the user has asked along with the patient responses; this feature is hidden by default.

After the encounter, a summary can be configured to display feedback of the user's actions, achievements, or mistakes. The system can also display a score and any other values the scenario designer wishes to track. This output display is controlled in the Ink file, which is interpreted and does not require programmatic change. These summary items are clickable and, if selected will take the user to an After-Action-Replay (AAR). The encounter's summary page is able to trigger an AAR when an individual summary entry is selected, or the Review Encounter button is pressed. The Review Encounter button triggers the AAR without passing any highlighted variables. However, clicking a summary entry can pass the AAR one or more variables to highlight—example variables are trust or user bias. The scenario designer can define any desired variable in the Ink file and the AAR system will show the user how their choices affected it or could have affected it. When viewing the AAR, a progress bar across the top of the screen contains a notch for every choice the user made. This bar is fully interactable and clicking a notch takes the user directly to that choice. The arrow buttons along the left-hand side of the screen are used to step though the user's choices one at a time. When a replay step is selected the choices available to the user at that time are displayed and the system automatically highlights the user-triggered button. If the AAR system is passed Ink variables to highlight and the user selected a choice that increased one of these variables, then that progress bar's notch is green. The notch is red if their choice decreased the variable, or if there was a choice that increased the variable, but the user chose a different option. If neither of these two previous situations applies, then the notch appears white.

For the interactive medical manikin, the system has the ability for the O/C to load previously generated Pulse scenarios via NL, which starts the patient in a particular physiological state with predetermined vitals. Furthermore, different Ink files define distinct scenarios, and their associated intents and expected actions. During this effort, medical SMEs and software developers collaborated to detail wounded soldier scenarios in which field care for a patient is provided. Scenarios were created with the Laerdal SimMan 3G advanced patient simulator in mind and available (see following figure), but nXcomms can be applied to myriad patient simulators. The physiology engine, the manikin, and software platform allow for nearly any scenario as well as administration of pharmaceuticals. Figure 1 shows the Laerdal SDK and interface to which the software connects. The Laerdal application provides a direct connection to the SimMan 3G manikin and shows the data being sent to and received from the manikin.

For the scenarios, it was supposed that the combat medic (CM) may have the items shown in Table 1 below at their disposal. The scenario begins with the user (a CM) arriving one to two minutes after an injury was sustained. The patient is calm and able to give answers but in obvious pain. The patient has a severe hemorrhage in the right leg and will develop a tension pneumothorax in the right lung. Resting vital signs are: blood pressure (BP) 120/70; heart rate (HR) 64 beats/min; respiratory rate (RR) 16 breaths/min; initial blood volume 5500 mL.

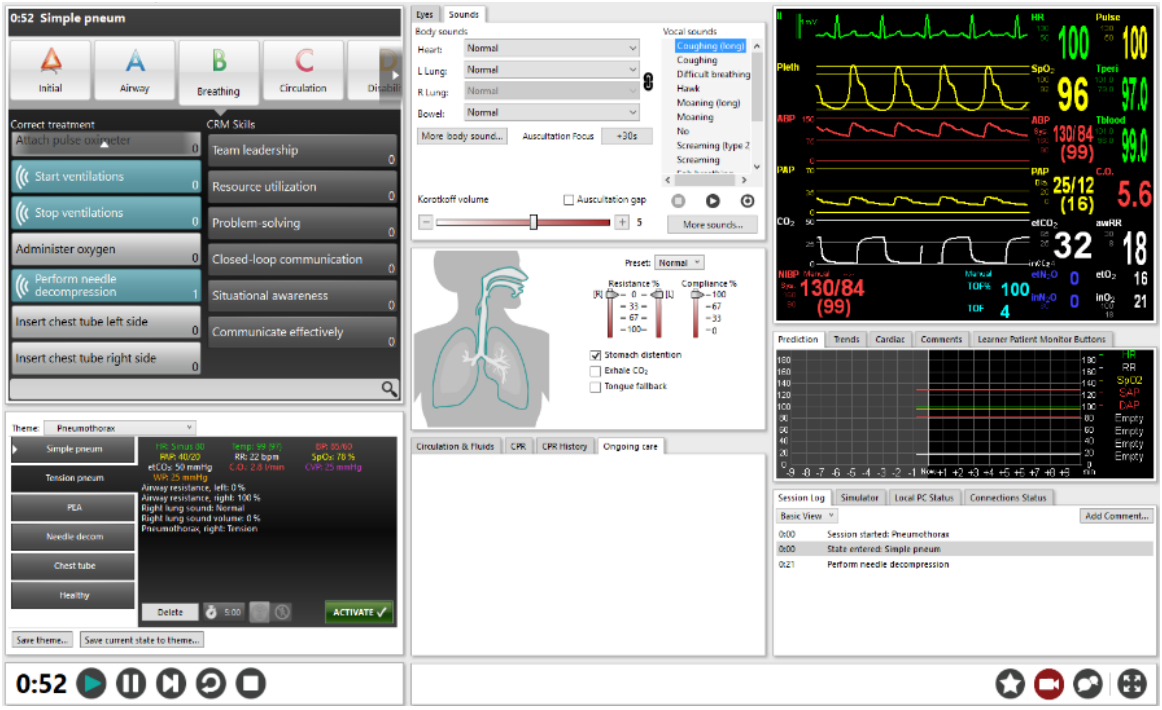

Figure 1. Visual representation of Laerdal SDK

For scenario part A, a 24-year-old male on patrol sustains shrapnel injuries from an IED explosion, causing a leg laceration. There is observable laceration of the right leg below the knee. A proximal (groin) tourniquet is applied. Medevac is unavailable within two hours. Vitals are: BP=144/84; HR=122 beats/min; RR=24 breaths/min. During secondary survey the CM is able to isolate all shrapnel to below the knee, requiring re-placement of tourniquet distally (just above knee) to preserve thigh tissue. The manikin/casualty (CAS) is laid on ground, with leg injury covered by the uniform. The uniform is bloody and a pool of blood around leg indicates blood loss. The O/C sets bleed rate of 85 mL/min due to external hemorrhage on right leg. The O/C informs the CM that CAS is awake/alert, and is actively trying to put a tourniquet on, though is unable to coordinate movement. This information tells CM that CAS has no significant spinal injury. The CM approaches and initiates dialog.

NL processing allows for branching flow to the progression of dialog, with changes to the exact language coded. The CM was encouraged to use typical speech and language; as described above, the system captures the intent of the dialog, which leads to appropriate responses; whereas the virtual patient has a branching conversation structure, this is a flat conversation meaning the user can ask any question at any time.

For scenario part B, a 24-year-old male on patrol sustains shrapnel injuries from an IED explosion, causing a leg laceration plus tension pneumothorax. Initial injuries are noted to the face, neck, and chest as well as a 4½ inch deep laceration to the right mid-thigh which is bleeding profusely. Vitals are: BP=114/83; HR=100 beats/min; RR=15 breaths/min (dips to 13); blood vol=~4600 mL. A hasty tourniquet is applied immediately to the right thigh. The CAS is on a bed with injuries exposed. The O/C sets bleed rate of 170 mL/min due to external hemorrhage on right leg. (Note: Femoral artery normally flows at ~150 mL/min, bumping up due to stress.) The O/C informs CM that CAS is awake/alert and has no significant spinal injury. The CM approaches and initiates dialog.

User Interface

The user experience for nXcomms was designed specifically for this project and contains some unique elements. A framework was implemented that creates asynchronous connection among the system, Pulse, and a Laerdal manikin. It is important to repeat that nearly any physiology engine and/or manikin could be substituted. The interface uses a local IIS web server and supports connections over WiFi. Several features are as follows: (i) push real-time state updates multiple times per second from Pulse to the manikin so that Pulse physiology simulation can be used to control all patient vital signs, with support for an adjustable timescale as appropriate for network and hardware performance; (ii) send Laerdal events to manikin (e.g., pause/resume simulation), which also gives the option of controlling all manikin features directly (see Table 2), including the vital signs if desired; (iii) receive event updates two ways from

Table 1. Combat medic scenario items available

Component	Drugs
Combat Pill Pack (contains three medications)	Tylenol (acetaminophen) 500 mg tablet (x2) Mobic (meloxicam) 15 mg tablet (1) Avelox (moxifloxacin) 400 mg tablet (1)
Pain medications (different dosing based on treating pain or for sedation)	Ketalar or Ketaset (ketamine) 500 mg per 5 mL (100 mg/mL) vial Dosing: IV/IO: 0.1-0.2 mg/kg push every 10-30 minutes [typical dose is 20 mg] IM: 0.4-0.8 mg/kg IM injection every 10-30 minutes [typical dose is 50 mg]
	Actiq (fentanyl) lollipop 800 mcg Dosing: typically one OTFC (oral transmucosal fentanyl citrate)
	Fentanyl citrate 250 mcg / 5 mL vial Dosing: typically 50 mcg IV/IO every 30-120 minutes as needed
	Percocet (acetaminophen/oxycodone) 5/325 mg tablet Dosing: typically 1-2 tablets every 4-6 hours as needed
	Morphine 10 mg/mL vial Dosing: typically 5-10 mg IV/IO every 1-6 hours as needed
Nausea/ vomiting	Zofran (ondansetron) 40 mg / 20 mL (2 mg/mL) vial Dosing: typically 4mg every 4 hours as needed

Table 2. Manikin control statements

Start hemorrhage, e.g., "Start hemorrhage at 50 ml per minute"
Increase/decrease hemorrhage by a fixed amount or by a percentage of the current value (e.g., "Decrease hemorrhage by 10%", "Reduce bleeding by 50 ml per minute")
Apply tourniquet
Apply/release pressure dressing
Insert tamponade (e.g., "user inserted tamponade reduce hemorrhage by 95%")
End hemorrhage
Inject morphine (e.g., "inject 5 ml of morphine subcutaneous at 1 mg per ml")
Start saline infusion (e.g., infuse saline intravenous 100 ml per sec with a bag volume of 1000 ml)
Remove saline infusion
Start tension pneumothorax (e.g., "start tension pneumothorax right lung severity 75%"
Increase/decrease tension pneumothorax by fixed percentages
Collapse lung
Needle decompression—support for O/C saying the command in case the manikin does not have the functionality to send this user action
Start/stop intubation
Apply oxygen (e.g., "start ventilating at 5 L per minute")
Listening to right/left or both lungs (e.g., "user listening to right lung")
Ability to record notes about the session (e.g., "Note user checked pupil responsiveness")

the manikin to cover the different ways a user and O/C can interact with the manikin; one system allows for registering a delegate that receives single updates from the SDK, and the other polls the SDK for batched updates, which are then parsed for the events to listen for; (iv) support for various physiology parameters sent to the manikin, including heart rate, blood pressure, SpO2, temperature, and others; (v) ability to send multiple physiology parameters in parallel to minimize needed bandwidth; (vi) support for manikin event updates sent in real time, so the O/C immediately sees changes to the manikin state; (vii) separate manikin event updates from physiology updates, reducing log traffic to relevant updates; (viii) login interface supporting multiple possible manikins and allowing the user to select and connect to the correct one; (ix) receiving events in the application when the user interacts with the patient manikin (e.g., by checking the pulse performing a needle decompression, inserting a chest tube; essentially anything the manikin can send); and (x) sending tension pneumothorax lung resistance and other manikin-specific data to the manikin. The ability for the O/C to use natural voice to talk to the physiology engine is present, via the NL processing module, to give the commands. Pulse engineers served as collaborators to incorporate additional required functionality, specifically with the added callbacks for internal physiology engine events (e.g., "saline solution bag is empty", "patient has entered hypoxia").

Overall UI capabilities include: (i) callbacks so that UI elements can be updated when physiology parameters change

(e.g., a change in the severity of the tension pneumothorax, or a hemorrhaging amount update); (ii) dynamic graphical representation of the current injury hotspots; (iii) display of multiple system events in the event log UI, including those received from the manikin, the physiology engine, and simulation; (iv) hooking up state UI elements to physiology/manikin events; (v) ability to pause/play the simulation; (vi) administering various medications, including Ketamine; (vii) blood transfusions; and (viii) internal hemorrhage support (i.e., spleen, liver, brain, aorta, splanchnic).

Field Demonstration

A technical demonstration was conducted on December 19, 2019 at Uniformed Services University of the Health Sciences (USUHS) in Bethesda, MD, presenting the following scenario: The warfighter receives a gunshot wound to the leg, thereby requiring the combat medic to control the bleeding by applying a tourniquet. After the warfighter has been stabilized, he is then transported via aeromedical ambulance to a Role 2 care facility. At altitude, the warfighter develops a tension pneumothorax, which is resolved by performing a needle decompression, thereby releasing the pressure from the pleural cavity and allowing heart and lung function to return to normal. During the technical demonstration, the control panel initiated the scenario, marked the exact times when the tourniquet and needle decompression procedures occurred, and signaled the scenario's conclusion. The Pulse physiology engine controlled the symptomology of both manikins. Finally, the software platform interface (Figure 2) visualized the patient's physiology parameters in real time and provided summary statistics at the scenario's conclusion.

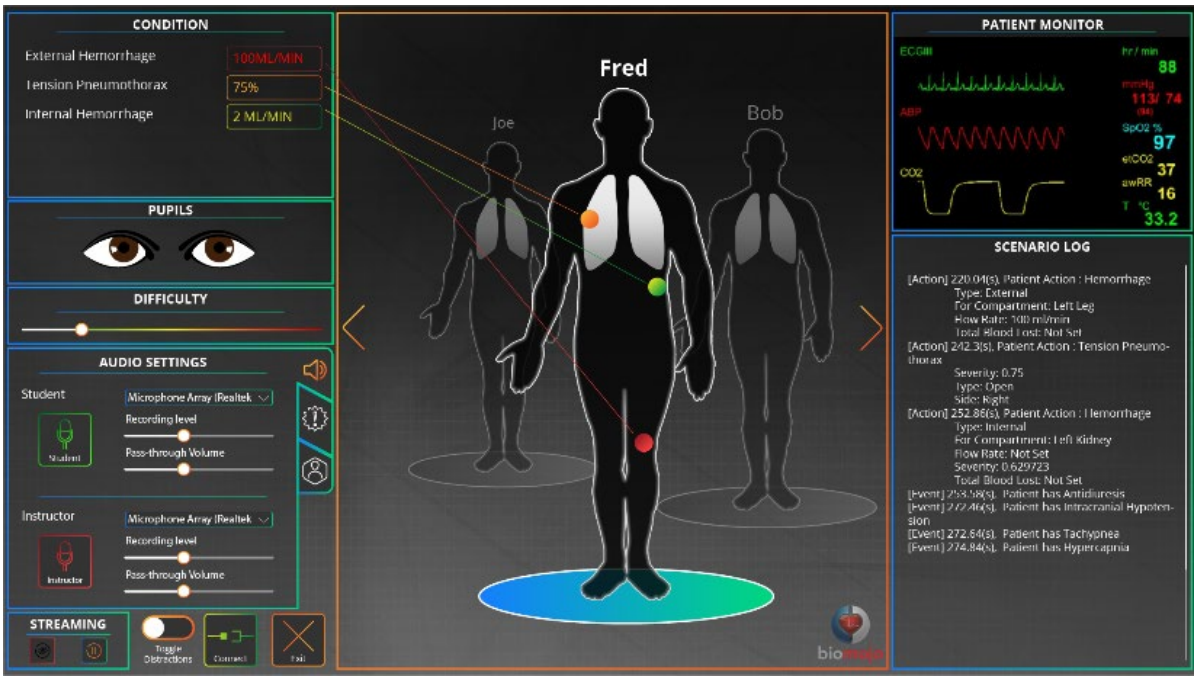

Figure 2. System interface

In the application and scenario, the top left displays the status of the scenario (e.g., amount of bleeding, severity of the tension pneumothorax, and whether a needle decompression has been attempted). This information is pushed from the Pulse engine to the Laerdal SDK so that the manikin reacts accordingly. The top right is the Pulse engine data, which is pushed to the manikin via the Laerdal SDK. So, if the Pulse engine specifies a respiration rate of 16, that state is reflected both on the manikin and in the Laerdal application. The bottom right of the application is the log of all events from the Pulse engine, the application, and the Laerdal SDK. When the system receives notification of a needle decompression event from the manikin, the notification appears here, and the system forwards the event to the Pulse engine. When the Pulse engine changes state (e.g., the patient enters tachycardia), the event appears, and the system pushes any relevant info to the manikin. The figure in the center of the image shows the location and severity of current injuries. In this image, the patient suffers from a tension pneumothorax and bleeding from the femoral artery. The bottom left corner of the application contains controls for the application: volume levels, microphone levels, etc.

The design allows the whole chain to be exercised either as part of a large training event, or in separate steps where each part of the care chain can be trained independent of the others. In the latter situation, the simulation starts from the saved state of the prior simulation; learners care for the patient as per normal and then save their state to a library.

That saved state can then be accessed and used as the starting point further along the care chain. The design meets the current medical simulation needs and is flexible to adapt to changing technology both in the underlying transport layers and in the higher-level application layers.

In the demonstration of nXcomms at USUHS, time was divided between the virtual patient and the Laerdal manikin portions. For the manikin, the SimMan 3G patient simulator was used (though the approach is vendor and device agnostic, to enable alternative manikins as well as virtual patients, in mixed reality holograms, and part-task trainers). The Laerdal SimMan 3G LLEAP SDK was licensed as were Laerdal's Patient Monitor Software and SimDesigner. The virtual patient relied on the nXhuman architecture developed originally at UNC (Hubal & Heneghan, 2017). The summary at the end of the virtual patient encounter was demonstrated to show what the clinician did correctly vs. incorrectly and to show a trust meter and score. Clicking on the trust meter brought up an AAR to review choices made during the encounter, allowing viewers to step through the encounter one question at a time or jump to any question by clicking on the progress bar at the top of the screen: a green tick indicated the trust value increased while a red tick indicated where it decreased. The ability for viewers to exit replay mode and continue the encounter from that point was also demonstrated.

SUMMARY AND FUTURE DIRECTION

This nXcomms effort involved: (i) improving language understanding through extended dialog scripting, phraseology collection and generation, and intent matching; (ii) the ability to capture simultaneous voice input as for patient and O/C dialogs; (iii) improving language generation through response selection, conversational fall-through, SSML metalinguistic controls, and keyword prioritization; (iv) performance improvement through optimization, caching, and memory usage reduction; (v) RESTful API to any external module to include Pulse physiology platform and Laerdal SimMan 3G; (vi) manikin control through voice command; physiology control through manikin interaction; (vii) scenario design with parameterized initialization, branching dialog, necessary interplay among connected components (AI modules, manikin), and assessment against learning objectives; and (viii) developing to be compatible with the DOD's Medical Simulation Training Architecture (Scheirich et al., 2019), which provides an open standard for military medical simulation to readily connect system components such as hands-on simulators, virtual patients, and performance measurement engines. Complex combat trauma scenarios were demonstrated using a manikin and a screen-based virtual patient. The software was designed from the ground up to be a manikin-agnostic product.

The software components are modular and interoperable. These capabilities were demonstrated by (i) integrating an off-the-shelf physiology engine developed by a partner, Kitware, (ii) integrating an off-the-shelf, widely used hands-on simulator from Laerdal, (iii) using publicly available software-as-a-service tools from Amazon and Google to manage critical components of NL understanding and generation, and (iv) preparing scenario content to use scriptable elements to control for initialization, dialog management, gesture and emotion expression, and performance assessment.

Looking forward, multiple directions are envisioned for improving language learning. For instance, to speed up the creation of the intents file involves building a database of questions (with their corresponding alternate ways of asking) that can be pulled from to quickly build future scenarios. A prototype website is able to make it easier to process matches and mismatches to build an intents file. And support is being added for automated synonym substitution, so that the system can capture how users ask the same question in different ways. Similarly, existing databases can be taken advantage of such as recordings from past medical encounters to which team members, in particular the UNC personnel, have access. These recordings are diverse and will serve as a basis for machine learning of the dialogs. In concert, a word-to-vector neural network is being integrated, incorporating the latest developments in linguistic semantics for NL processing. These models are shallow, two-layer neural networks that are trained to reconstruct linguistic contexts of words. The neural net takes as its input a large corpus of text and produces a vector space, typically of several hundred dimensions, with each unique word in the corpus being assigned a corresponding vector in the space. Word vectors are positioned in the vector space such that words that share common contexts in the corpus are located in close proximity to one another in the space. Relatedly, a tool is being developed to automate the implementation of the branching conversation scripts into Ink without the additional step involving a dialog developer. The concept is to learn from encounters; that is, as a given user's input is captured, it is then used to inform the NL for the next user, and so on. Structures are in place, to include the specific intents file that is used to 'understand' the string of words that is captured by Google, modified manually now but through a process to be automated. For patient response, future versions will also be able to modulate speed and tone of voice according to patient stress level, and separately by generating dynamic responses based upon previously recorded actor voices, thus producing more nuanced human like responses. Additionally, the system is designed to run optimally via a cloud connection. Internal experiments were conducted with the system running offline, resulting in much less precise and slower simulated

patient responses. Much work needs to be done to push this capability farther.

Last, the concept of what it means to present a virtual patient is being extended by incorporating additional AI modules. For example, the patient's personality and emotions are continually being improved. A scenario designer can give the virtual patient a specific personality, or define emotions and/or traits to track within the Ink script. In the virtual patient encounter, trust and a patient bias value increase or decrease based upon the questions the user asks. These values are then used to both change the virtual patient's demeanor and how they respond to future questions. Similarly, in the future, details will be added on how conversations flow, using existing research into scientific argumentation and rhetoric. The idea is to build upon models of conversational flow that take characteristic or prototypical shape. Also, cognition and metacognition model additions to the architecture will enable the virtual patient to reason about responses and reactions and their appropriateness based on rules derived from literature and validated through observation. Finally, adding to the existing after-action review capability by incorporating user modeling will allow for tracking variables across sessions (so that the virtual patient would remember how that clinician treated them previously, as in cases where the clinician has multiple encounters with the same patient) and introduce difficulty and complexity into the scenario to better gauge the learner's competency.

Potential DOD customers include facilities that have manikin based and/or screen-based patient simulators in their simulation center inventories. These include the U.S. Army Medical Simulation and Training Centers, the Army Central Simulation Committee medical treatment facility (MTF) based sim centers, the Navy and Air Force Medical Modeling and Simulation Training MTF based centers (NMMAST/AFMMAST), the joint Medical Education and Training Center (METC) campus and USUHS. Tactical Combat Casualty Care (TCCC), Prolonged Field Care (PFC), Prolonged Casualty Care (PCC), and Advanced Cardiac Life Support (ACLS) medical training scenarios (e.g., hemorrhage control, airway compromise, extremity fractures/traumatic amputations, penetrating trauma to chest and abdomen with/without tension pneumothorax, fasciotomy, closed head/traumatic brain injury, concussion with/without loss of consciousness, ocular injuries) should be used to challenge the capability of the platform, either manikin or screen based. These injuries are very difficult to train a live action role player (LARP), for learners to understand and apply techniques correctly, so that the system may have tremendous impact on training realism.

ACKNOWLEDGEMENTS

This work was supported by the U.S. Army Medical Research Acquisition Activity, award #W81XWH-19-C-0119. The views expressed are those of the authors and do not reflect official policy or position of the U.S. Army.

REFERENCES

Bray, A., Webb, J.B., Enquobahrie, A., Vicory, J., Heneghan, J., Hubal, R., TerMaath, S., Asare, P., & Clipp, R.B. (2019). Pulse physiology engine: An open source software platform for computational modeling of human medical simulation. SN Comprehensive Clinical Medicine, 1(5), 362-377.

Guinn, C., & Hubal, R. (2003). Extracting emotional information from the text of spoken dialog. In Proceedings of the Workshop on Assessing and Adapting to user Attitudes and Affect: Why, When and How? Johnstown, PA.

Hubal, R., & Heneghan, J. (2017). Carolina virtual patient initiative [abstract]. Pharmacy Education, 17(1), 292.

Kizakevich, P., Furberg, R., Hubal, R., & Frank, G. (2006). Virtual reality simulation for multicasualty triage training. Proceedings of I/ITSEC (pp. 170-177).

Morbini, F., et al. (2012). A mixed-initiative conversational dialogue system for healthcare. Proceedings of the Annual Meeting of the Special Interest Group on Discourse and Dialogue (pp. 137-139).

Scheirich, H., Beaubien, J., Metoyer, R., De Novi, G., & Kelliher, T. (2019). Toward the development of a medical simulation training architecture (MSTA). Proceedings of I/ITSEC (#19219).

Sottilare, R., Hackett, M., Pike, W., & LaViola, J. (2017). Adaptive instruction for medical training in the psychomotor domain. Journal of Defense Modeling and Simulation: Applications, Methodology, Technology, 14(4), 331-343.

Taylor, G., et al. (2012). A multi-modal intelligent user interface for supervisory control of unmanned platforms. Proceedings of Conference on Collaboration Technologies and Systems. IEEE.

W3C. (2010). Speech Synthesis Markup Language (SSML) Version 1.1. https://www.w3.org/TR/2010/REC-speech-synthesis11-20100907