Amplifying Human Ability through Autonomics and Machine Learning in IMPACT

Iryna Dzieciuch*, John Reeder, Robert Gutzwiller, Eric Gustafson, Braulio Coronado, Luis Martinez, Bryan Croft, Douglas S. Lange
Space and Naval Warfare Center Pacific, San Diego, CA, USA 92152-5001

ABSTRACT

Amplifying human ability for controlling complex environments featuring autonomous units can be aided by learned models of human and system performance. In developing a command and control system that allows a small number of people to control a large number of autonomous teams, we employ an autonomics framework to manage the networks that represent mission plans and the networks that are composed of human controllers and their autonomous assistants. Machine learning allows us to build models of human and system performance useful for monitoring plans and managing human attention and task loads. Machine learning also aids in the development of tactics that human supervisors can successfully monitor through the command and control system.

Keywords: Machine Learning, Task Management, Plan Monitoring, Autonomics

1. INTRODUCTION

Command and control has always been the practice of directing teams of autonomous systems, whether those systems were individual marines, ships controlled by humans, or unmanned aircraft controlled by biologically inspired artificial intelligence. However, the employment of systems without immediate human control, has led to an impedance mismatch of sorts between the commander and the commanded, increasing the complexity of the command and control problem along two dimensions. First, the cognitive load of understanding the tactical situation is stressed by the impedance mismatch, and second, because we can more easily add large numbers of systems to the environment.

Utilizing a combination of autonomics and machine learning, the Intelligent Multi-UxV Planner with Adaptive Collaborative Control Technologies (IMPACT), enhances the human’s ability to exercise control over forces by discretizing and managing the task space, monitoring performance against plans, and learning tactics. The autonomics framework is designed to flexibly manage networks. Within IMPACT, we define networks of operators and their autonomous assistants as well as defining plans as networks of activities. We then utilize the autonomics framework to manage the networks.

Machine learning provides models of behavior when we don’t want or cannot adequately model those behaviors by hand. To create tasks, rules or human input can be used. However, we also attempt to learn how to create tasks by examining past human-specified tasks. Models of human performance in executing tasks are learned in order to predict future performance on similar tasks as well. Finally, machine learning is used in a unique manner to learn behaviors for autonomous systems such that human controllers can recognize correct and aberrant behaviors reducing the impedance mismatch between the artificial intelligence and the human intelligence in command.

2. TASK MANAGEMENT IN IMPACT

2.1 Challenges

The IMPACT scenario is reliant on the smooth interaction between autonomous systems and a human supervisor. This interaction generates a large number of tasks to be completed by the user of a C2 station. Due to the high volume of tasks, tasking can easily become overwhelming and disorganized, leading the user to become frustrated and less effective in completing assigned tasks. Addressing these issues inspired the development of a management system capable of determining user tasks, dividing tasks into a hierarchy, presenting tasks to the user, and providing a mechanism to execute an action for each task.

*iryna.dzieciuch@navy.mil; phone 1-619-553-2793

Micro- and Nanotechnology Sensors, Systems, and Applications IX, edited by Thomas George, Achyut K. Dutta, M. Saif Islam, Proc. of SPIE Vol. 10194, 101941Y · © 2017 SPIE
CCC code: 0277-786X/17/$18 · doi: 10.1117/12.2262849

2.2 Task Generation

Determining an instance in which a task must be created requires the Task Manager to employ IMPACT's networking hub. Tasks can be generated in a variety of ways. For instance a query can be sent to a user regarding an asset or a UAV can fly into a restricted area. Every event in IMPACT is forwarded through the hub. By parsing chat and notification messages that pass through the hub, we ascertain information for generating tasks. Task generation follows the subsequent pattern:

- A chat message is received from a designated room in the IChat repository.
- The contents of the message are compared to a map of regular expressions.
- Each regular expression pattern provides an associated task definition.
- If the message matches to a regular expression pattern, a task is instantiated by providing the task definition. This task is referred to as the parent task.
- Any subtasks associated with the parent task definition are assigned to the parent task

Keywords in the chat messages help to determine the task category which in turn helps to determine the play type. For instance, if a chat message is received from by the Task Manager with the following text, "Unidentified watercraft heading towards the shore (Boat Golf) at 30.427560, -87.145746," a task is provided for the user to "Provide Overwatch." This task can be completed by selecting a subtask to, "Call Point Search," or to "Surveil Watercraft." The current categories of tasks include intruder events, environmental events, vehicle failures, base defense events, Random Antiterrorism Measures (RAMs) and Queries. Other important information gathered from the task generation include the time the task was assigned, the time the task was completed, the sequence number of the task and the priority associated with the task category.

A secondary method used to generate tasks is to parse message headers as messages pass through IMPACT's networking hub using the ZeroMQ protocol. This method is employed for tasks generated by the occurrence of constraint and Restricted Operations Zone (ROZ) violations. In these cases, the message origin does not come from a chat room, but directly from the IMPACT hub. Messages from the hub are parsed and filtered to search for a specific header, "json:ML:Global.Notify". When these messages are discovered, a task is generated in a similar manner to the pattern outlined above.

2.3 Play Calling

An essential function of the Task Manager is allowing the user the ability to call a play from a task listed in the Task Manager. Each task assigned to a user in an IMPACT scenario may require the execution of a play for itself or a subtask in order to be completed. A task may have multiple play options available for execution. Many plays, such as queries, have a single associated option. Other plays, however, may have multiple methods in which it can be assigned. The plays associated with each task are determined by a Quick Reaction Checklist. When the user selects a play, the Task Manager will auto-populate a workbook that can be used to execute the chosen play. Play options are gathered from the metadata of the task and the workbook is spawned using IMPACT's internal workbook spawn mechanism.

2.4 Autonomous Assistant and Load Balancing

The latest innovations incorporated into the Task Manager involve helping the user to balance their work load by tasking an autonomous assistant with extraneous tasks. The autonomous assistant provides services to receive tasks and execute them. So far, much has been done to alleviating operator tasking on lower priority or repetitive tasks, but these tasks may increase in importance or rarity in the future.

Currently, the autonomous assistant is able to be tasked in two ways. The operator can manually assign tasks to the autonomous assistant by simply clicking a button. Tasks can also be reassigned to the operator by selecting the task from the Autonomous Assistant's task list.

The autonomics framework is utilized by modeling the operators and their intelligent assistants as a network of servers. We are managing the task queues, and use repair strategies when the network gets bogged down. These strategies can include moving tasks from one server to the other (e.g., moving from the human to an autonomous assistant), or using a different algorithm for distributing new tasks (e.g., increasing the preference for having an autonomous assistant perform future tasks). Critical to making these decisions are learned models of the human performance on the types of tasks being queued, and also on establishing the working agreements created between humans and the autonomy [18].

3. LEARNED TASK GENERATION

As discussed above, we can generate tasks for the user from rules. Simple patterns in the text of chat messages can be utilized to recognize the need for a task. But many tasks remain outside the system, in the human's brain. For task management to work, these tasks must be made into discrete data objects within the system. There are many ways of eliciting this knowledge from users that may also be useful. In the current approach, however, we look mainly at the data in the system.

All data in the IMPACT simulation is stored in states, which store the live data for all of the vehicles in the simulation. Other data comes from the sensors of the UxVs. This data is stored in camera, video stream, and radio state variables. Each vehicle state is comprised of many variables such as: current location, velocity, acceleration, current heading, available energy, energy usage rate, list of payloads, and current tasks. All of this data can be used by machine learning techniques to determine what play should be generated.

In using the task manager for IMPACT, a user can create their own play for a job at any point in time. The play could be of any type, and the user can create new types of tasks previously unknown to the system. The trigger for the creation of the tasks can therefore be based on available information in the simulation. We have no prior knowledge about which data in the simulation was used to make the task; however, further generation of domain knowledge would be a useful for human-computer interaction comprehension. It could be based on any or all information in the system. Therefore, the initial machine learning approach taken is the K-Nearest Neighbor algorithm (kNN) [1] based on its simplicity and applicability to many problems.

Our high-level approach is comprised of three steps:

1. Record the states of the IMPACT simulation when a task is created.
2. Continuously monitor the IMPACT simulation states.
3. If the current simulation state matches a state previously recorded when a task was made, then generate this task for the user of the Task Manager.

The core of kNN requires us to determine the distance metrics between the states of the simulation in order to match them and ultimately generate a task. This is not a trivial task as the states of the vehicles in the simulation rarely ever match exactly. The feature standardization technique [2] is used to remove any bias caused by the state variables having different units and thus different measurement scales.

Initial implementations of the approach have highlighted a number of important design questions: Should the variables in each state in the distance measurement be weighted evenly? Are vehicles considered to be homogenous and thus their states can be compared to another vehicles' state? Should the distance measurements for all individual states be combined or can subsets of states be compared? Finally, what is the threshold for similarity between states for a task to be generated? These questions require optimizing the way of optimizing data – user interactions.

3.1 Task Optimization

In this section we will discuss the complexities of the IMPACT system which can result in the human operator suffering information overload. It calls for an optimization model that monitors queuing of tasks in IMPACT with the aim of reducing operator's cognitive load.

All vehicle raw data collection from the IMPACT system yields two types of data: user generated chat messages and sensor data. Messages from the user come in the form of natural language and can be broken down into four elements: time of message, location, duration and asset. These representations of data are summarized in Table 1.

Time	*Point (ROI)*	*Duration (time)*	*Asset (Sensor, UxV)*
00:00:00	Ammo Dump	40 min	Imagery
00:17:00	Chow Hall	10 min	360 degrees (imagery)
20:00	Gate 2	10 min	Force (vehicle)
30:00	Gate 2	5 min	Force (vehicle)

Table 1 Example of the key information obtained from chat messages

The Time column in Table 1 represents the time stamp when the message was issued or when the action is planned to take place. The Point column represents the region of interest (ROI) where assets are planned to appear. Duration is the time of job completion when the trigger is lifted. Asset is the name of the sensor or vehicle that is the capability which is used to complete a task. The idea behind structured chat message retrieval from a database is to quickly distinguish important semantics from the natural language into computer-readable form. This stage requires further development and will not be covered in this report.

For the user who simultaneously controls a number of autonomous agents, complexity means higher rates of multitasking. In this case, complexity reduction happens when many simpler tasking states are grouped together in a relevant sequence.

Timing also plays an important role, as events occur at a random rate. For example, the tasking system can be doing a routine task, such as ground monitoring, but if we introduce additional information or events, the user's attention will be diminished. Because accumulation of simple tasks at a high rate is overwhelming, timing and rate of task occurrence in the user task queue is a key factor.

Environmental events take place outside of the user's control. These events trigger a user's reaction which will require actions in the IMPACT system to respond to the environmental events. Example environmental events include: Gate runner, Mortar fire, and a user's observation of a chat message. These events have pre-programmed scenario plays and quick reaction responses that are evoked and monitored by user in the IMPACT system.

Some of the variables in the IMPACT system include data that is supplied by a sensor from the unmanned vehicles (UxV). UxV's operate in a time and space domain and carry variable sensor performance characteristics, for example: Airspeed, Energy Rate, Altitude, and Latitude/Longitude coordinates, etc. The user is constantly updated with sensor information as he or she performs data retrieval when a chat message query is issued. The data representation is summarized in Table 2 below, but it is also composed of the time and ROI information, as well as the duration for the task completion, UxV status, sensor status, and vehicle status.

Time	*Point (lat/long/alt)*	*Duration (time)*	*Sensor Characteristics*

00:00	30.471585;
	87.181458;
	650
		0	Airspeed: 300
			Energy rate: 100
			Pitch Angle: 0.5
			Sensor Type: IR camera

50:00	30.446;
	87.150706;
	0
		50/100 complete	Airspeed: 23
			Energy rate: 98
			Pitch Angle: 0.5
			Sensor Types: IR camera

Table 2 Example of UxV sensor data in the IMPACT system.

3.2 Play Calling

Play calling is a method that the user performs to control unmanned vehicles in the IMPACT system. It usually consists of a pre-programmed number of simpler actions that run their course, and are monitored by the user. User tasks are either triggered by a chat message or selected among a list of suggestions.

Common attributes of the data presented in the summary above are space and time. Both the sensors and the IMPACT operators see information in the space and time domain. All events and tasks occur at specific ROI and a point in time. Suppose, the basic problem we are considering is the state space S of time and location of all variables: sensor, chat (user) and environmental events. The control space is composed of the sequence of decisions in the tasking domain C. The data-task optimization problem of the IMPACT system can thus be stated as follows: What is the least complex sequence of tasks that needs to take place to satisfy success of the outcome within a specified completion time?

We can represent state space for the chat message variable as X = {x1, x2, x3, x4}, where x1 is time, x2 is point, x3 is duration, and x4 is asset. Similarly, Y = {y1, y2, y3, y4} is the sensor data with similar arguments. Given that currently, one can control a number of UxVs, the representation of these can be written down as X1 and Y1 for UxV1. The events that take place during a scenario can be represented as E = {e1, e2, e3, e4} that describes time, location, duration, and type of event. These events trigger a sequence of suggested tasks C that the user can do to reach a favorable outcome under different levels of complexity. For the purpose of illustration for the variable complexity tasking, we propose to use three main variations of complexity settings, as high, medium, and low. For example: High complexity is when there are 5 or more events in the tasking queue that require user attention, medium is when there are 3 events that requires user attention, and low is when there are less than 3 events that requires user attention.

For example, sequence of tasking decisions C by operator after a single event E is the following:

1. Send 2 vehicles with cameras to investigate damage - States: X1, Y1, Y2.
2. Send emergency vehicles to point Alpha - States: X1, Y2.
3. Inspect closed roads for blockages or obstacles - States: X1, Y2.
4. Continue until next message - States X2.

In the occurrence of a single event, the set of rules is straightforward but in the occurrence of simultaneous events the situation becomes rather complex. Imagine the user's reaction to multiple emergencies. That is why it is essential to be able to distinguish and prioritize a sequence of decisions or tasks in light of different complexity levels. The control problem can be formulated as follows: Optimize the use of the information available in space and time to find such a sequence of decisions C that gives the maximum result (successfully completed tasks) under varying complexity levels (CL) low, medium and high. Thus, the main problem of how to properly control and evaluate states under different complexities becomes a minimization-maximization problem.

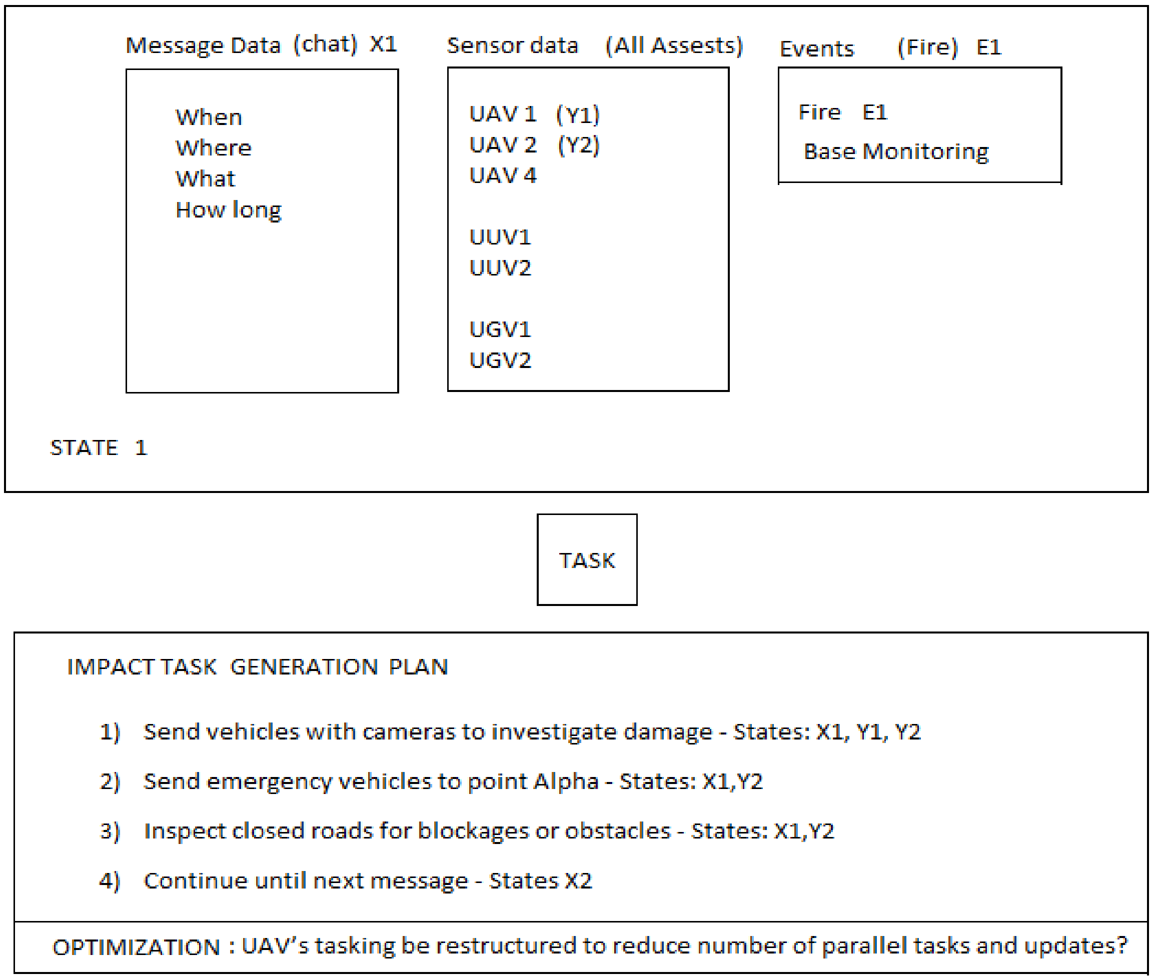
The proposed IMPACT planning problem can be generalized to the optimal selection of tasks max C, given a message state space X, and sensor data Y and event E, for a complexity level CL. The optimal scheduling and queuing of tasks is meant to reduce (minimize) complexity in the time and space domain. Mathematically it can be described as:

$$\min CL(t,s)\ [\max C(t,s)\,(X1,\ldots,Xn) \cup (Y1,\ldots,Yn) \cup (E1,\ldots,En)]$$

Thus, decision trees have been realized for the IMPACT system by creating subcategories. The repetitive decision for variable complexity can be described structured in the decision table. For example, are there any chat messages and/or sensor data in the 10-minute window period that overlaps in the time and space domain? How does this information influence the tasking state? Can an UxV's tasking be restructured to increase the number of parallel tasks and updates? Thus, we need to construct the decision table for each case, an example is shown in Table 3.

Time	*Space*	*UxV1*		*UxV2*		*Ground Patrol 1*		*Ground Patrol 2*	
		X1	*Y1*	*X2*	*Y2*	*X*	*Y*	*X*	*Y*
00:10	Position 1	X						X	X
00:14	Position 2		X	X					
00:21	Position 3					X	X		

Table 3. Example decision table. X and Y represent chat and sensors which provide data at certain time intervals in the simulation.


Message Data (chat) X1

When
Where
What
How long

Sensor data (All Assests)

UAV 1 (Y1)
UAV 2 (Y2)
UAV 4

UUV1
UUV2

UGV1
UGV2

Events (Fire) E1

Fire E1
Base Monitoring

STATE 1

TASK

IMPACT TASK GENERATION PLAN

1) Send vehicles with cameras to investigate damage - States: X1, Y1, Y2
2) Send emergency vehicles to point Alpha - States: X1,Y2
3) Inspect closed roads for blockages or obstacles - States: X1,Y2
4) Continue until next message - States X2

OPTIMIZATION : UAV's tasking be restructured to reduce number of parallel tasks and updates?

Figure 1 Model for decision making process data-user actions

After constructing an optimized tasking table we can observe behavior of the separate UxVs and their task load in the time and space domain. Upon closer examination, we observe that two chat messages sent at time 00:10 involve UxV1 and one Ground Patrol 2. At time 00:14 (or 4 seconds later) sensor data from UxV1 and message at UxV2 is sent during Task1. Now we can group chat messages data and sensor data that happens at the same time or are close in space. Next, we repeat the same procedure for consecutive Task 2. Clustering the sensor and chat message status is the first step that can be taken towards reducing the operational complexity for the user and to investigate if such data can be used to model the control system under different complexity levels. Such an approach, we believe, will help to optimize tasking decisions and reduce complexity for the human operator. Some of the clustering techniques, will help us understand the dynamics of task creation and understand how machine learning on state-space is used by task management.

4. AUTONOMICS FOR PLAN MONITORING

During an IMPACT scenario, there will be several active plays. This necessitates live monitoring of plan progress and anomalies. Vehicle telemetry, tasks and other plan elements must be monitored and used to evaluate holistic plan health. In addition to plan health, global constraints must be monitored and reported as alerts to the operator. Thresholds for acceptable plan and global values must be established and a mechanism to detect and take action upon breach of threshold must exist.

4.1 Autonomics

Plan Monitor leverages the Rainbow Autonomics Framework [3] developed at Carnegie Mellon University which enables autonomous management of networks through:

- The ability to dynamically monitor and analyze a network's properties.
- The ability to detect breaches on a network's architectural design assumptions.
- The ability to effect changes on a network in response to breaches in design assumptions.

Rainbow grants software architects the ability to establish a model of a network through the use of Acme, a formal architectural design language. Acme model components reflect network properties and Acme model rules enforce a network's design assumptions. Rainbow's probes and gauges collect data from the network to update the model dynamically while strategies, tactics and effectors provide automated adaptation to effect changes on the network. A key observation in IMPACT is that the structure of a plan allows us to represent plans as networks thereby granting us the ability to manage plans with Rainbow.

4.2 Plan Monitor Software Architecture

Plan Monitor is an external module in the IMPACT system. It is an extension of the Rainbow framework and is written in the Groovy programming language. Plan Monitor communicates with IMPACT through the network hub using ZeroMQ. Its software elements are as follows:

- Model
  - Establishes components representing air, surface and ground vehicles, tasks, areas of interest and zones (including restricted operating zones).
  - Establishes components representing plans by connecting the components above to reflect plan structure.
  - Establishes templates, rules and thresholds ensuring the integrity of plans.
- Probes
  - Subscribe to the network hub and ingest relevant messages.
  - Report data to the appropriate gauges.
- Gauges
  - Update components and properties in the model. May provide additional processing of data.
  - Employ mechanism to dynamically generate model components from network hub messages.
- Strategies and Tactics
  - Call effectors to take action upon detection of poor plan health.
  - Call effectors to take action upon global constraint breaches.

- Effectors
    - Publish plan health.
    - Publish constraint violation notifications.

4.3 Plan Health

The primary function of Plan Monitor is providing plan health information to the operator. Through the constant monitoring and evaluation of plans we can communicate their real-time status. Status falls within three categories: Nominal (Green), Lower Caution (Yellow) and Upper Warning (Red) with extent of deviation correlating to severity of status. Plans generally have two phases that Plan Monitor must consider:

- En-route – Vehicle has been assigned to a plan and is on its way to its destination. Parameters include:
    - Fuel – Thresholds are set by templates in the model for each vehicle type.
    - Speed – Thresholds are determined by plan metadata – each plan includes a set of way points for vehicles to follow with each way point establishing expected vehicle speed.
    - Expected Time to Execute (ETE) – Thresholds are cached upon the instantiation of a plan by using the distance between way points and expected vehicle speed. Real-time vehicle telemetry is compared to its expected position along the route establishing ETE quality.
- On-task – Vehicle has reached its destination and is performing a task. Parameters include:
    - Fuel – Same as above.
    - Speed and Task Quality – Thresholds are determined by the task associated with the current plan.

On-task plan health is determined by the type of tasks in a play. Although there are over twenty available plays, they follow six patterns. In order to generalize the calculation of plan health, these patterns are used to categorize plays into plan types as follows:

1. Search Plan – Involve vehicles focusing their cameras on points or lines in the world.
2. Watch Plan – Involves vehicles focusing their cameras on a vehicle.
3. Escort Plan – Involves vehicles maintaining a distance from a vehicle.
4. Cordon Plan – Involves vehicles maintaining a distance from a point in the world.
5. Blockade Plan – Involves vehicles maintaining a distance from a point in the world.
6. Comm-Relay Plan – Special case as this plan is not called but generated to support a called play.

While the details involved in these patterns may vary (friendly vs non-friendly vehicle for watch plans), it is sufficient to measure on-task plan health. For multi-vehicle plans, health for each vehicle is compared and the lowest quality parameters are combined into a single plan health update. In the current version of Plan Monitor, we have the ability to detect when a plan is performing poorly and communicate this to the IMPACT system. However, using this same detection mechanism, it is possible to invoke mission-correcting strategies such as cancel, re-plan or request a change to a poorly performing play.

4.4 Global Constraints

A secondary function of Plan Monitor is effecting the IMPACT scenario through notifications. Currently, there are three types of notifications:

1. Fuel Notifications. A rule is established in vehicle templates in the model with each vehicle type defining its fuel threshold. A low fuel notification is published upon threshold breach.
2. ROZ Notifications. A rule is established in a strategy triggered upon the generation of a ROZ Violation component in the model. A ROZ violation notification is published upon vehicle or way point presence in ROZ.
3. Flight line Notifications. A rule is established in a flight line template in the model with location of flight line and vehicle response time thresholds. A flight line violation notification is published when there are no vehicles within time thresholds.

These constraints are constantly evaluated at all times, regardless of whether there are any on-going plays.

4.5 Component Generation

Since an IMPACT scenario can be complex with multiple types of plays, manually developing models for each case is a non-trivial effort. To that end, Plan Monitor employs a method of generating model components and connections between components dynamically. Using the Java reflection API, LMCP object metadata read from the hub is used to generate corresponding structures in the model.

An LMCP object to model conversion follows this pattern:

1. LMCP object is read from the hub.
2. Reflection tools collect field names, types and values (including those of parent classes in the object's class hierarchy)
3. Presence of an abstract model component (template) for that object's class is verified and generated if it does not exist. (This operation happens once per object type.)
    a. Presence of a concrete model component (instantiation) for that object's instance is verified and generated from a template if it does not exist. (This operation happens once per unique object instance.)
4. Component is updated using object's field values if the values are different.

This pattern allows us to generate a model of any plan developed in IMPACT.

4.6 Customized User Interface and Working Agreements

Due to the generic qualities of Rainbow, its built-in user interface (UI) provides tools useful to developers of Rainbow applications. However, the end-user is likely not concerned with the information presented by these tools as they communicate actions relating to Rainbow’s internal processes. This provides an opportunity to explore a customized user interface with utility relevant to the network it is managing.

The implementation of our customized UI is through a Rainbow gauge. This gauge initializes the UI and employs a visitor software design pattern to establish itself as a UI controller. UI components are tied to settings in the model and changes are communicated through the gauge. This link between UI and model allows us to control strategy selection by means of rule conditions. Consider the following rule applied to vehicle templates:

```
rule fuelRule = invariant !GUI_ALLOW_FUEL_UPDATES or EnergyAvailable > FUEL_CRITICAL;
```

Here we ensure that vehicle fuel is above a threshold and observe how the GUI_ALLOW_FUEL_UPDATES property affects the rule. This property is tied to a checkbox component in the Working Agreements UI section. If the checkbox is unchecked, the property value is false and disables the rule despite EnergyAvailable falling below threshold. Thus, disabling strategies tied to vehicle fuel status by means of the UI.

This mechanism supports a goal in our work with autonomics which is to promote transparency and collaboration between human machine teams. Through working agreements we establish a policy dictating what the autonomy is allowed to do (but also what the human agrees to do and handle). A motivating factor behind this effort is the realization that a human operator may have critical information about the world that the autonomy does not. Therefore, the ability to control the level of automation on demand is valuable.

5. HUMAN AUTOMATION INTERACTION

IMPACT as a whole is focused on tools and autonomy to aid a human supervisor in managing large teams of autonomous vehicles. Autonomy will help the supervisor resource their plays, and to plan the routes for the vehicles. The underlying autonomy of the vehicles is assumed to be robust and efficient. This assumption however, is in doubt as we move towards a future where swarms, and dynamic environments require the use of machine learning techniques to develop the underlying autonomy of the vehicles. As the number of vehicles grows the workload on an individual operator will become ever more burdensome. To alleviate this, more robust autonomy needs to be developed, especially in the case of large numbers of drones acting in concert, such as a swarm. Often pure machine learning techniques can produce efficient behaviors, but those behaviors might seem foreign to the supervisors who must make the decision on whether to allow the vehicles to continue to operate.

We are elsewhere [19] exploring an Interactive Machine Learning approach to advance algorithms that benefit from the best of both worlds; human-supervisable but optimized algorithms. Naturally, these investigations take place for human-robot interactions, because of the dangers posed by physical agents operating at odds with human expectancies [4]-[6]. Although military command and control (C2) tasks are good candidates for autonomy and machine learning integration, they also are steeped in tradition and human jargon, including concepts like commanders intent [7]. Opaque agent behavior will naturally disrupt C2; but interactive machine learning for unmanned robotic behaviors may bridge this gap. Without some shared understanding, machine learning will be difficult to supervise [8], at odds with the move to make systems transparent and trust-able to users [9]-[14]. Neuroevolutionary computation [15]-[17] carries such a downside, despite other obvious benefits.

Despite our plans, additional work remains to validate and explore the approaches already described, including the user of task and play monitoring displays. We intend to conduct human-in-the-loop experiments to assess their value in supervising autonomy.

REFERENCES

[1] E. Fix, J. L. Hodges, "Discriminatory analysis, nonparametric discrimination: Consistency properties," *Technical Report 4, USAF School of Aviation Medicine, Randolph Field , Texas*, 1951.

[2] L. E. Peterson, "K-nearest neighbor," Schoarepedia, 4(2):1883, 2009.

[3] E. Fix, J. L. Hodges, "Discriminatory analysis, nonparametric discrimination: Consistency properties," *Technical Report 4, USAF School of Aviation Medicine, Randolph Field , Texas*, 1951.

[4] V. Groom and C. I. Nass, "Can robots be teammates? Benchmarks in humanrobot teams," *Interact. Stud.*, vol. 3, no. 2007, pp. 483–500, 2007.

[5] J. R. Lee and C. I. Nass, "Trust in computers: The computers-are-social-actors (CASA) paradigm and trustworthiness perception in human-computer communication," in *Trust and technology in a ubiquitous modern environment: Theoretical and methodological perspectives*, D. Latusek and A. Gerbasi, Eds. Information Science Reference, 2010, pp. 1–15.

[6] C. I. Nass, B. J. Fogg, and Y. Moon, "Can computers be teammates?," *Int. J. Hum. Comput. Stud.*, vol. 45, pp. 669–678, 1996.

[7] R. Willard, "Rediscover the art of command and control," *Proc. - United States Nav. Inst.*, vol. 128, no. 10, pp. 52–54, 2002.

[8] T. B. Sheridan and R. Parasuraman, "Human-Automation Interaction," *Rev. Hum. Factors Ergon.*, vol. 1, no. 1, pp. 89–129, Jan. 2005.

[9] T. L. Sanders, T. Wixon, K. Schafer, J. Y. C. Chen, and P. Hancock, "The influence of modality and transparency on trust in human-robot interaction," *IEEE Int. Inter-Disciplinary Conf. Cogn. Methods Situat. Aware. Decis. Support*, pp. 156–159, 2014.

[10] J. D. Lee and K. A. See, "Trust in automation: Designing for appropriate reliance," *Hum. Factors*, vol. 46, no. 1, pp. 50–80, 2004.

[11] T. Sanders, K. E. Oleson, D. R. Billings, J. Y. C. Chen, and P. a. Hancock, "A model of human-robot trust: Theoretical model development," *Proc. Hum. Factors Ergon. Soc. Annu. Meet.*, vol. 55, no. 1, pp. 1432–1436, Sep. 2011.

[12] C. D. Wickens, J. Hollands, S. Banbury, and R. Parasuraman, *Engineering psychology and human performance*, 4th ed. Upper Saddle River, NJ: Pearson, 2013.

[13] R. Parasuraman and V. Riley, "Humans and automation: Use, misuse, disuse, abuse," *Hum. Factors*, vol. 39, no. 2, pp. 230–253, 1997.

[14] A. Freedy, E. DeVisser, G. Weltman, and N. Coeyman, "Measurement of trust in human-robot collaboration," *Proc. Int. Conf. Collab. Technol. Syst.*, 2007.

[15] J. Gauci and K. O. Stanley, "Generating large-scale neural networks through discovering geometric regularities," *Proc. 9th Annu. Conf. Genet. Evol. Comput. - GECCO '07*, pp. 997–1004, 2007.

[16] K. O. Stanley, D. B. D'Ambrosio, and J. Gauci, "A hypercube-based encoding for evolving large-scale neural networks.," *Artif. Life*, vol. 15, no. 2, pp. 185–212, Jan. 2009.

[17] K. O. Stanley and R. Miikkulainen, "Evolving neural networks through augmenting topologies," *Evol. Comput.*, vol. 10, no. 2, pp. 99–127, May 2002.

[18] R.S. Gutzwiller, S.H. Espinosa, C. Kenny, and D.S. Lange, "A Design Pattern for Working Agreements in Human-Autonomy Teaming", Appl. Hum. Factors Ergon. 2017.

[19] R.S. Gutzwiller and J. Reeder, "Human Interactive Machine Learning for Trust in Teams of Autonomous Robots", IEEE CogSIMA, 2017.