Human-Agent Teamwork in Cyber Operations: Supporting Co-Evolution of Tasks and Artifacts with Luna

Larry Bunch[1], Jeffrey M. Bradshaw[1], Marco Carvalho[2], Tom Eskridge[1], Paul J. Feltovich[1], James Lott[1], and Andrzej Uszok[1]

1. Florida Institute for Human and Machine Cognition (IHMC), Pensacola, FL, USA
2. Department of Computer Science, Florida Institute of Technology, Melbourne, FL, USA
{lbunch, jbradshaw, teskridge, pfeltovich, jlott, auszok}@ihmc.us
mcarvalho@fit.edu

Abstract. In this article, we outline the general concept of *coactive emergence*, an iterative process whereby joint sensemaking and decision-making activities are undertaken by analysts and software agents. Then we explain our rationale for the development of the Luna software agent framework. In particular, we focus on how we use capabilities for comprehensive policy-based governance to ensure that key requirements for security, declarative specification of task-work, and built-in support for joint activity within mixed teams of humans and agents are satisfied.

Keywords. Task-artifact cycle, coactive emergence, software agents, multi-agent systems, policy, Luna, KAoS, OWL, ontologies, cyber security

1 Introduction

Despite the significant attention being given the critical problems of cyber operations, the ability to keep up with the increasing volume and sophistication of network attacks is seriously lagging. Throwing more computing horsepower at fundamentally-limited visualization and analytic approaches will not get us anywhere. Instead, we need to seriously rethink the way cyber operations tools and approaches have been conceived, developed, and deployed.

Though continuing research to improve technology is essential, the point of increasing such proficiencies is not merely to make automated tools more capable in and of themselves, but also to make analysts more capable through the use of such technologies. To achieve this objective, we must adopt a human-centered approach to technology development [1].

In a memorable encapsulation of themes relating to human-centered technology development, Ford, *et al.* [2] argued that the accumulated tools of human history could all profitably be regarded as orthoses—not in the sense that they compensate for the specific disabilities of any given individual, but rather because they enable us to overcome the biological limitations shared by all of us. With reading and writing anyone can transcend the finite capacity of human memory; with a power screwdriver anyone can drive the hardest screw; with a calculator, anyone can get the numbers right; with an aircraft anyone can fly to Paris; and with IBM's *Watson*, anyone can beat the world Jeopardy champion. Eyeglasses, a familiar instance of an "ocular orthosis," provide a particularly useful example of three basic principles:

- *Transparency.* Eyeglasses leverage and extend our ability to see, but in no way model our eyes: They don't look or act like them and wouldn't pass a Turing test for being an eye. Moreover, eyeglasses are designed in ways that help us forget we have them on—we don't want to "use" them, we want to see *through* them.
- *Unity.* Since our goal is not making smart eyeglasses but rather augmenting vision, the minimum unit of design includes the device, the person, and the environment. This mode of analysis necessarily blurs the line between humans and technology.
- *Fit*. Your eyeglasses won't fit me; neither will mine do you much good. Prostheses must fit the human and technological components together in ways that synergistically exploit their mutual strengths and mitigate their respective limitations. This implies a requirement for rich knowledge not only of technology, but also of how humans function.

Orthoses or prostheses are useful *only* to the extent that they "fit" — in fact, the "goodness of fit" will determine system performance more than any other specific characteristic. This is true whether one considers eyeglasses, wooden legs, or cognitive orthoses. One can identify two broad categories of fit — *species fit* and *individual fit*. In some cases, a particular aspect of human function can afford a consistent fit across most of a population of interest. In many other instances, however, an *individual fit* is desirable, and in these cases relevant differences among individuals must be accommodated [3]—and adjusted as needed.

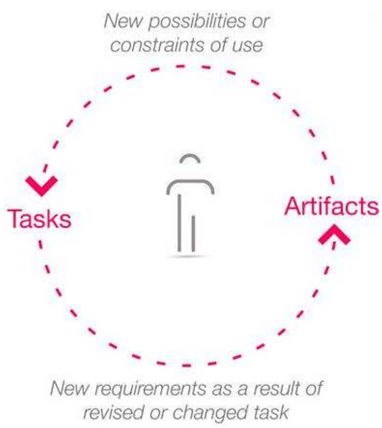

Fig. 1. The Task-Artifact Cycle.

The term "fit" is pertinent for an additional reason: it evokes the concept of evolution—specifically a *co-evolution* of the user *task* and the technology *artifact*. This concept of a co-evolution of tasks and artifacts is not new, but goes back two decades to an influential chapter by Carroll, *et al.* [4]. The task-artifact cycle includes two phases: the first involves the design and development of artifacts to help users perform their assigned tasks; the second concerns the way that the use of the artifacts defines new perceptions, possibilities, or constraints of use that change the way the task is performed (see fig. 1).

This never-ending cycle of co-evolution between mutually dependent tasks and artifacts is an inevitable challenge to software developers—though, on the bright side, it does provide job security. On the user side, however, it means that the capabilities of software will always lag behind current needs, particularly in domains such as cyber operations where the nature of threats is constantly changing.

In this article, we describe an agent-based approach to the support of co-evolution of tasks and artifacts to improve human performance while increasing transparency, unity, and fit. Though we will discuss cyber operations as an application example, we believe that the same principles and capabilities we outline here will be useful in many similar domains. Section 2 outlines the general concept of *coactive emergence* as an iterative process whereby joint sensemaking and decision-making activities are undertaken by analysts and software agents. Section 3 describes the background and motivation for our Luna agent framework, focusing primarily on its policy-based features that support the requirements for security, declarative specification of taskwork, and built-in support for joint activity within mixed teams of humans and agents. Section 4 will give an overview of the KAoS policy services framework. Section 5 will briefly describe our objectives in the design of Luna agent framework, and selected highlights of its features. Section 6 will illustrate by example some of the ways in which policy governance is being exploited within Luna. Finally, section 7 will outline current areas of research and future directions.

2 Coactive Emergence

Coactive emergence describes an iterative process whereby secure system configurations, effective responses to threats, and useful interpretations of data are continuously developed through the interplay of joint sensemaking and decision-making activities undertaken by analysts and software agents [5]. The word "coactive" emphasizes the joint, simultaneous, and interdependent nature of such collaboration among analysts and agents. We will illustrate how this applies to cyber sensemaking (fig. 2):

1. Analysts gather evidence relating to their hypotheses through high-level declarative policies that direct and redirect the ongoing activities of agents.
2. Within the constraints of policy, agents manipulate system configurations to improve security and intepret the data, optionally enriching their interpretations through machine learning techniques. Because of their built-in abilities to work together dynamically to analyze and synthesize meaningful events from the raw data, agent interpretations can be more easily made to match the kinds of abstractions found in human interpretations more closely than in other approaches.

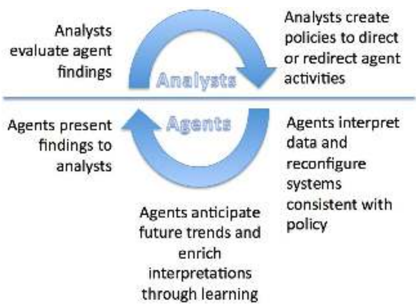

Fig. 2. The Coactive Emergence Cycle.

3. Agents aggregate and present their findings to analysts as part of integrated graphical displays, and analysts interact with

these displays in order to explore and evaluate how agent findings bear on their hypotheses.

4. Based on refinements of their hypotheses and questions from these explorations and evaluations, analysts redirect agent activity as appropriate.

The process of “emergence” operates at two levels:

- First-order patterns emerge from agent and analyst interpretations of data that are shaped by problem-space constraints currently expressed within policies and tool configurations. For example, through the application of analyst-defined policy-based pattern recognition, agents may tag and display selected network data as instances of emergent threats. Likewise, a display of current agent results may lead analysts to recognize the possibility of additional emergent threat patterns that the agents may have missed.
- Second-order emergence arises from dynamic changes made by agents and analysts to the problem-space constraints. For example, analysts may add, delete, or change agent policies in order to refine their data interpretations or their responses to threats. Agents may also change their own policies through policy learning. When permitted, agents may also propagate learned policies to other agents.

3 Background and Motivation for the Luna Agent Framework

IHMC’s innovations in research and development of software agent frameworks for multi-agent systems stretch back more than fifteen years (see, e.g., [6, 7, 8]). As a happy consequence of progress in the field since those early days, there now exists a range of interesting agent frameworks, serving different purposes, available from a variety of commercial sources and research institutions, that can be applied with confidence to many practical applications. For this reason, we had not expected that we would ever have to create a new agent system. However, contrary to expectation, we have recently developed a new agent framework called *Luna*, named for the founder of Pensacola, Tristán de Luna y Arellano (1519 – 1571). In this section of the paper, we attempt to explain the reason why.

The short answer is that when we were confronted with the need to apply software agent technology within the domain of cybersecurity operations, we found that no current platform adequately met our needs. Foremost among our requirements were the following three requirements, which we will discuss one by one.

In considering the *security requirements* of currently available software agent platforms, the key role of policy constraints that could govern behavior at every level of the agent system readily became apparent. In our previous experiences with agent systems, we had discovered that people sometimes were reluctant to deploy agents exhibiting significant autonomy because of concerns that the freedom and indeterminacy of such agents might allow them to do undesirable things [9]. We have found that the ability to use policy to impose constraints on agent behavior, in essence providing a guarantee to people that agent autonomy would always be exercised within specified bounds, gave people the assurance they needed to feel that highly capable agents could act in a trustworthy, predictable, and safe manner.

Second, with respect to the need for a platform supporting the interactive *formulation of common agent tasks by end users*, rather than by software developers, we believe that policy systems may also prove useful. In past experience with military, space, and intelligence applications, we have learned that many common tasks can be formulated as declarative obligation policies that require given actions to occur when triggered by a specified context. Further enhancing the usefulness of such capabilities to define agent taskwork is the potential for specifying abstract obligations (i.e., "goal" policies or "collective obligations") on an initially-unspecified population of agents in a fashion that is similar to how concrete obligations are imposed on specific classes and instances of agents.

Third, to satisfy the need for a platform that would provide built-in support for effective *coordination of joint activity within mixed teams of humans and agents*, we believe that a policy-based approach also provides a viable option. Based on our research and development experience in a variety of applications involving the coordination of human-agent-robot teamwork (HART), we believe that important aspects of teamwork such as observability, directability, interpredictability, learning, and multiplicity can be addressed by policy-based mechanisms [10, 11].

In the following section, we describe the KAoS policy services framework that provides the basis for the governance of the Luna platform and its agents.

4 The KAoS Policy Services Framework

Because agents are powerful, we use powerful policy management and enforcement frameworks to govern their actions. Whereas many special-purpose policy approaches are optimized only for specific kinds of tasks (e.g., access control) and support only the ability to permit or forbid an action, the ontology-based approach of KAoS enables semantically-rich specifications of virtually any kind of policy. It supports not only the ability to permit or forbid an action in a given context, but also to require that certain actions be performed when a dynamic, context-specific trigger is activated (e.g., start doing X, stop doing Y, reduce your bandwidth usage, log certain actions)—or to waive such an obligation dynamically if the situation warrants.

The KAoS Policy Services framework [12] was the first to offer an ontology-based approach (based on the W3C standard, OWL 2 (http://www.w3.org/TR/owl2-overview/) to policy representation and reasoning. It is currently the most successful and mature of all such efforts. Following collaborative efforts by the NSA-sponsored Digital Policy Management (DPM) Architecture Group and IHMC, the KAoS core ontology was adopted as the basis for future standards efforts in DPM [13].

KAoS has already been integrated into IHMC's Luna agent framework, as well as several other agent platforms and traditional service-oriented architectures. Preliminary work has been done on agent learning mechanisms that propagate learning with localized opportunistic mechanisms inspired by biological analogues. In addition, we plan to develop capabilities for KAoS to take advantage of localized agent learning results by allowing new policies to be constructed programmatically, with optional human oversight. This would allow learning results from groups of individual agents that are of high generality or urgency to be rapidly propagated to whole classes of other agents.

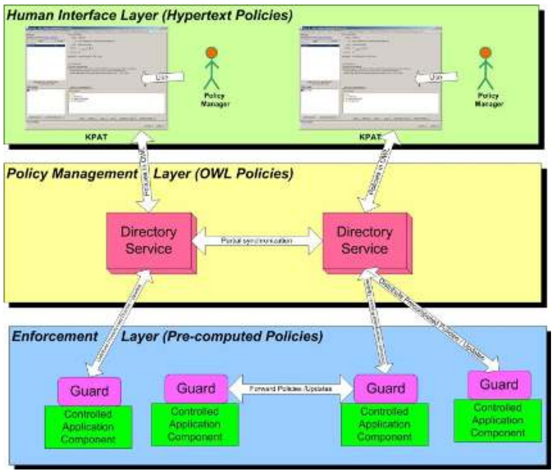

Fig. 3. Notional KAoS Policy Services Architecture.

Two important requirements for the KAoS architecture are modularity and extensibility. These requirements are supported through a framework with well-defined interfaces that can be extended, if necessary, with the components required to support application-specific policies. The basic elements of the KAoS architecture are shown in . Within each of the layers, the end user may plug-in specialized extension components if needed. KAoS also contains capabilities for dealing with a variety of policy conflicts and for analyzing and testing policies.

KAoS ensures that the Luna agents respect all security and privacy policies, that they respond immediately to human redirection, and that they have the teamwork knowledge they need to work with analysts and other agents collaboratively. KAoS policies also ensure that the entire system adapts automatically to changes in context, environment, task reprioritization, or resources. New or modified policies can be made effective immediately.

5 The Luna Agent Framework

5.1 Overview of Luna Features

In our cybersecurity applications, Luna agents function both as interactive assistants to analysts and as continuously-running background aids to data processing and knowledge discovery. Luna agents achieve much of their power through built-in teamwork capabilities that, in conjunction with IHMC's KAoS policy services framework, allow them to be proactive, collaborative, observable, and directable. Luna also relies on KAoS for capabilities such as registration, discovery, self-description of actions and capabilities, communications transport, and messaging.

Figure 4 shows how KAoS integrates with Luna to provide services and to enforce policies. An OWL representation of Luna is maintained within the KAoS Distributed Directory Service. Through its interactions with the Luna host environment, KAoS

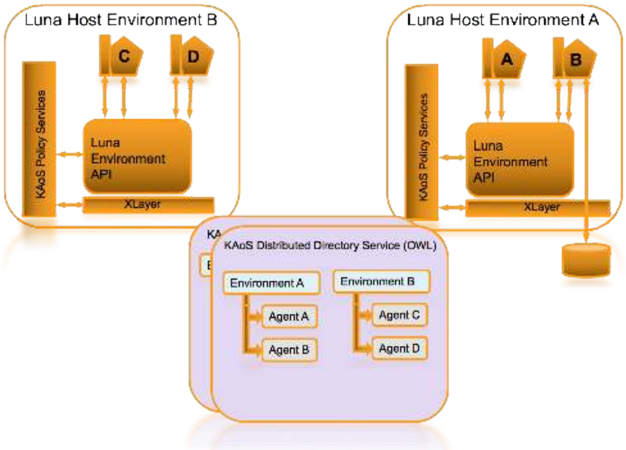

Fig. 4. Luna Conceptual Architecture.

regulates the lifecycle of both the environment (e.g., start and stop Luna) and the agents (e.g., create, pause, resume, stop, and move agents). Policy can also regulate environment context for shared agent memory (e.g., getting and setting its properties), allowing efficient parallel processing of large data sets. An agent-based implementation of context mirroring across different Luna environments is provided. Through policy, the Luna host environment also governs agent progress appraisal—a subject to which we will return later.

Because Luna policy checking and enforcement is done by virtue of KAoS-based "method-call" messages to agents and other components, actions taken by an agent on its own (invoking its own methods) are not subject to policy. This design choice posed no problems for the use cases we envisioned.

In the future, KAoS will also integrate with VIA to provide a means of policy enforcement outside the Luna host enironment. VIA [14] is a next generation cross-layer communications substrate for tactical networks and information systems. VIA allows fine-grained enforcement of policy down to operating-system-level operations such as the opening of a socket and the monitoring and filtering of specific elements of agent messaging.

In order to support dynamic scalability, load balancing, adaptive resource management, and specific application needs, the Luna platform supports the policy-governed option of allowing the *state* of agents (vs. *code* of agents) to migrate between operating environments and hosts. The Luna environment maintains agent mailboxes with message forwarding when agents migrate. Luna state mobility will provide the foundation for future implementation of agent persistence (i.e., saving and loading agent state to a persistent store).

5.2 Automated Mapping Between Java and OWL

Within the base class for Luna cyber agents are defined some common agent tasks that can be called through OWL descriptions. However, one of the most important innovations in Luna is the ability to add custom agent actions to the policy ontology, based on their Java equivalent. IHMC provides a Java2OWL tool to assist with this task.

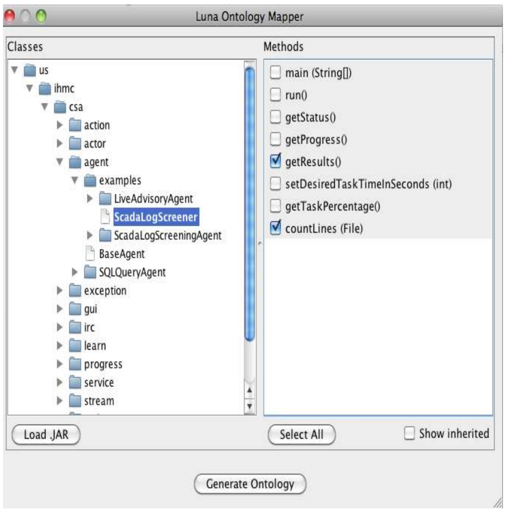

Fig. 5. Java2OWL Tool.

To understand this feature, it is important to understand that the framework allows translation from the KAoS 'method call' messages to OWL action instance descriptions for policy checking, and back to method calls. The ability to convert an OWL description to a Java method call is the feature that puts 'teeth' in obligations. A Luna environment can invoke such methods on itself or any set of agents hosted on that Luna, or pass the obligation to one or more remote Luna environments for execution. Combine this obligation invocation feature with the fact that obligations can be triggered by one set of actors and fulfilled by another set of actors (e.g., Luna obligated to do X when Agent does Y, All Agents obligated to do X when Luna does Y), and we have a foundation for the implementation of what are called in KAoS "collective obligations" [12]. These are obligations that specify *what* must be done by some group of agents without saying exactly *how* it should be done and *which* specific agent(s) should do it.

The Java2OWL tool (fig. 5) can be used to browse custom agent code, select methods to bring under policy control, and generate an OWL description for the selected method signatures. These methods are then available for policies as Actions performed by Agents of that type. The only prerequisite is that agent code must be available (on the classpath) when Luna starts.

Our ontology for method call requests is currently low-level, representing Java Methods and their parameters including the parameter data types and the order of the parameters in the method signature.

5.3 Selected Applications of Luna to Cyber Operations

Agents play a variety of roles in Sol, our cyber operations framework [15]. Among the most demanding is in multi-layer agent processing and tagging of live or retrospectively played-back NetFlow data representing worldwide Internet traffic. A high-level view of roles and relationships among agents relating to these functions is shown in figure 6.

Incoming UDP traffic goes to a NetFlow agent for parsing and transformation into Java objects (1). In principle, the same or different data could be routed to multiple NetFlow agents on the same or different hosts to share the processing load. The NetFlow agent sends the data to any number of Tagger agents that work in parallel in real-time to tag the data (2). For example, Watchlist agents tag data that appears on

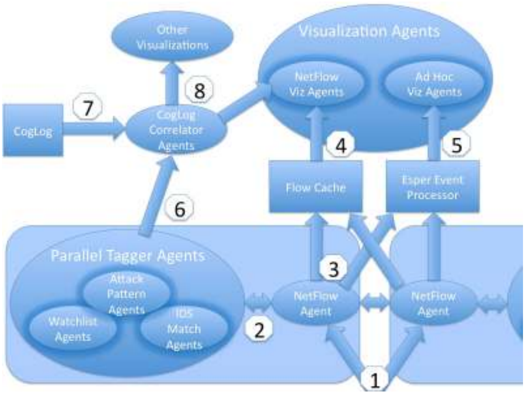

Fig. 6. Agent processing and tagging of NetFlow

whitelists or blacklists while IDS Match agents tag data corresponding to intrusion detection alerts. Drawing on selected results from low-level tagging agents, Attack pattern agents may be defined to look for higher-level attack patterns. By this means, agent annotations do not merely highlight low-level indicators of threat patterns, but can directly identify the type of threat itself. For instance, instead of requiring the analyst to notice that a configuration of connecting lines (some of which may be obscured) indicates a distributed port scan, agents working on abstracted data semantics can directly indicate the source of the attack. As another example, if a message is anomalous because it is sending oversized packets to a port associated with an SQL database, higher-level agents can abstract that message and represent it as an instance of an SQL injection attack. A system of semaphores ensures that all the Tagger agents have completed their work before the NetFlow agent sends results to the Flow Cache (3). NetFlow Visualization agents enforce policies that mediate data being sent to analyst displays, ensuring, among other things, that data not authorized for viewing by particular users are automatically filtered out (4).

The Esper complex event processor [16] provides support for efficient ad hoc queries of many types that can be initiated and consumed by other visualization agents (e.g., Stripchart View agent) or by agents of other types for further processing (5). We are also considering the use of Esper for data stream handling further upstream in the agent analytic process.

CogLog Correlator agents ingest combined data from selected Tagger agents operating on real-time data (6) and historical data within the CogLog (7). The CogLog is a Semantic-Wiki-based tool prototype with which software agents and human analysts can maintain and use a log of findings pertinent to a given investigation, while also linking to other relevant information from prior cases. Unlike the real-time Tagger agents, the Correlator agent can perform deeper kinds of analytics in “out of band” mode. Among other things, this correlated information can help different analysts “connect the dots” between related investigative efforts. The Correlator agents may send additional data annotations to NetFlow Visualization agents and/or to agents supporting other visualizations (e.g., Connection Graph view) (8). Our Attack Pattern Learning Agents provide another example of an “out of band” agent type. These agents consume and process all NetFlows (rather than just subsets of tagged data produced by Tagger agents) in order to learn and propagate useful threat patterns.

In the future, exploration of larger questions of adversarial intent, attack strategies, and social connections among attackers could also proceed along similar lines of increasing abstraction in agent processing. The ability to reduce perception and reasoning requirements on the analyst through fixed or ad hoc organizations of agents processing visual and logical data dimensions is a major benefit of agent-based analytics.

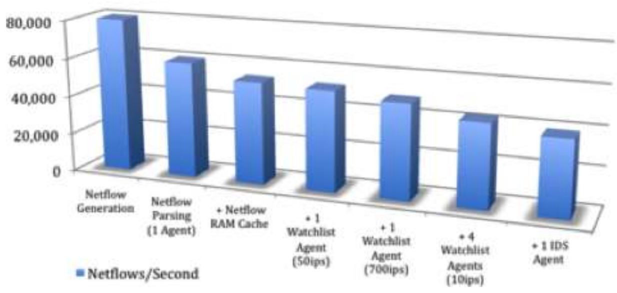

Fig. 7. Initial Luna performance data.

Initial performance data on Luna is promising, even though we have not yet focused significant attention on optimization of the framework. With respect to live processing on our test configuration (Mac Pro with two Quad-core Intel Xeon @ 2.26GHz with 16 GB RAM, 1000baseT Ethernet, Mac Pro RAID Level 0, 4x2TB), the IHMC network of ~100 nodes is the only one we have tested thus far. Performance at this small scale of less than 1000 flows/second is, of course, excellent.

With respect to retrospective performance in our current configuration, the maximum rate of our CyberLab NetFlow emulator playing back Internet 2 data is 80k flows/second (~14MB/second) and the maximum rate of Luna agent NetFlow parsing is 60k flows/second. Sample configurations that include the additional task of maintaining a cache of NetFlows in shared RAM result in rates of 52k flows/second (single watchlist agent with 50 ips on its list), 49k flows/second (a Watchlist agent added with 700 IPs on its list), 43k flows/second (four more Watchlist agents added with 10 IPs each on their lists), and 39k flows/second (an IDS Match agent added whose performance is constrained by file I/O).

We realize that these performance numbers fall short of requirements for some large-scale cyber applications. We are confident that an effort to optimize agent processing within and between agents would yield significant performance increases. More importantly, because of the distributed nature of the Luna architecture, we are in a good position to take advantage of whatever scale of computing and network resources might be available for a given application.

6 Luna Policy Examples

Luna is governed by policy statements that take either the form of authorization or obligation policies as follows:

- Actor is [*authorized* | *not authorized*] to perform Action requested by Actor with attributes…
- [*before* | *after*] Actor performs Action requested by Actor with attributes, Actor is [*obligated* | *not obligated*] to perform Action with attributes…

When Luna policies are defined, the underlined terms above (Actor, Action) are replaced in point-and-click fashion with more-specific concepts automatically drawn from the extensible ontologies maintained within KAoS. Actors in the policy statements above could be made to refer to one or more classes or instances of the Luna host environment, e.g.:

- Class: *All Luna environments*
- Extensionally defined collection of groups: *Luna in group 'NOC_A', 'NOC_B'*
- Intensionally defined collection of groups: *Luna with context property 'Alert Level' in ('Critical', 'High')*
- Extensionally defined collection of individuals: *LunaNOC_A_1, LunaNOC_A_Shared*
- Intensionally defined collection of individuals: *Luna containing agent 'BotnetC2Correlator'*

Actors could also be made to refer to classes and instances of Luna agents, e.g.:

- Class: *All Agents*
- Intensionally defined Group: *Watchlist matching agents*
- Extensionally defined Group: *Agents in group NOC_A*
- Extensionally defined Group: *Agents running in Luna NOC_A_1*
- Specific Instances: *BotnetC3Correlator_Agent, ZeusWatchlistAgent*

To make these ideas more concrete, we now give two groups of examples: 1). Teamwork policies; 2). Network Operations Center scenario policies.

6.1 Teamwork Policy Examples

It is one thing to enable software agents to perform the taskwork needed to help human analysts with complex high-tempo tasks and quite another to equip them with the capabilities that allow them to become good team players. Our perspective on resilient human-agent teamwork comes from joint activity theory [17], a generalization of Herbert Clark's work in linguistics [18, p. 3]. Our theory describes the criteria for joint activity, outlines aspects of its "choreography," and highlights the major requirements for effective coordination in situations where the activities of parties are interdependent. For the purposes of this discussion, we focus primarily on examples of the sorts of policies we are designing to support human-agent teamwork, under the headings of *observability, directability, interpredictability, learning,* and *multiplicity*:

- *Observability*: An important aspect of observability is being able to know how agents are progressing on their current tasks, especially those in which their actions are interdependent (e.g., "I'm not going to be able to get you the information you need to get started on your task by the deadline to which we agreed previously.") To support this, we have implemented built-in mechanisms and policies for *progress appraisal* [19] in Luna.
- *Directability*: When agents need to be redirected due to changes in priorities, new knowledge, or failures, users can add and retract obligations on particular agents or classes of agents or Luna environments at runtime. This includes obligations relating to life-cycle control, such as pausing or resuming their operation.

- *Interpredictability*: One way in which the interpredictability of an agent can be assessed is through a combination of data on its current progress with its past history of work in similar contexts.
- *Learning*: The observability features of agents can be used to support capabilities for policy learning—i.e., creating new KAoS policies programmatically based on patterns that are consistent and important to tasks being undertaken by a whole class of agents. The process of learning itself may be subject to policies relating to the scope of adaptation permitted in a given context. It may also be subject to optional policy requirements for human oversight.
- *Multiplicity*: Multiplicity, the requirement for multiple perspectives on the same data, can be supported by policy-based enforcement of data consistency across these perspectives. For example, policies would ensure that changes in one view of the data would propagate correctly (and with the appropriate policy restrictions on what can be viewed) to other kinds and instances of views on that data.

Of the areas mentioned above, progress appraisal and agent (re-)directability through obligations are currently the most well-worked-out aspect of these five human-agent-teamwork-based considerations in Luna. As an illustration of how these considerations can be supported through policy, we describe our implementation of progress appraisal below.

Providing regular reports of agent progress is an integral feature of every Luna agent. The Luna environment handles all of the progress management including:

- Registration and deregistration of users and agents to receive progress reports from particular agents;
- Maintaining a timer to send agent progress reports periodically (e.g., every minute);
- Querying the agent periodically for its current progress, or providing an interface for agents to announce milestone-based progress events;
- Distributing the agent's progress reports to the interested parties.

The decision to have the Luna environment manage progress appraisal rather than relying on the agents themselves was a deliberate one. Some of the key advantages over agent self-managed progress appraisal include:

- Luna can provide progress in conditions where the agent cannot;
- Luna may pause an agent so that it would no longer capable of sending progress messages.
- The agent may be buggy or otherwise unresponsive, but Luna will still send progress to users (indicating that the agent is unresponsive);
- Policy control over the frequency and recipients of progress appraisal enables directing or redirecting progress appraisals from groups of agents to other agents for further analysis.

6.2 Network Operations Center Scenario Policy Examples

In the development of experimental scenarios for network operations center use of our framework, we considered requirements for access control, action authorization, information sharing, and coordination between local and remote sites. Below we give some illustrative examples of KAoS policy support in Luna for these issues.

Imagine a scenario involving two cooperating network operations centers (NOC) at different locations, NOC_A and NOC_B. Each NOC has its own policies, in addition to policies that are shared between them.

NOC_A has three Luna environments:

- *Luna_NOC_A_Monitoring*: Within this environment, monitoring administrators from NOC_A create and maintain agents to support shared visualizations.
- *Luna_NOC_A_Analysis*: Within this environment, analysts from NOC_A create agents to perform ad hoc analysis and investigation tasks.
- *Luna_NOC_A_Shared*: Within this environment, analysts from either NOC_A or NOC_B can create agents to perform ad hoc analysis and investigation tasks.

NOC_B has one Luna environment:

- *Luna_NOC_B*: Within this environment, analysts from NOC_B create agents to perform monitoring, ad hoc analysis, and investigation tasks.

KAoS uses the concept of "domains" to define the scope of policies. In this case, the two NOCS will share a domain. In addition, each NOC will have its own domain, and, within NOC A, each NOC A Luna environment will be a subdomain to the NOC A domain. For the convenience of the administrator wanting to minimize the number of policies that need to be defined, the mode of a domain can be set to be "tyrannical" (where everything is forbidden that is not explicitly authorized) or "laissez-faire" (where everything is permitted that is not explicitly forbidden). Here are some examples of policies in the scenario, assuming a tyrannical domain.

Authorization Policy Examples. This positive authorization policy specifies that NOC Administrators can make any request of any Luna environment:

Any Luna is authorized to perform any Action
requested by a member of the NOC_Administrators Group

This positive authorization policy allows any user to make requests of any Luna environment belonging to the same group as that user.

Luna in any Group is authorized to perform any Action
requested by a member of the same Group

The positive authorization policy permits remote users from NOC_B to manage agents within the shared Luna environment, while the negative authorization policy prevents these users from lifecycle actions such as stopping the environment or changing its context properties:

LunaNOC_A_Shared is authorized to perform any Action
requested by a member of Group NOC_B

LunaNOC_A_Shared is not authorized to perform any environment lifecycle action
requested by a member of Group NOC_B

Obligation Policy Examples. This positive obligation policy requires any newly created Watchlist agent to send progress reports to the Watchlist Correlation agent:

After Any Luna performs create agent of type 'Watchlist Agent,' that Luna is obligated to add agent progress listener where:
listener is 'Watchlist Correlation Agent'

agent is the agent that was created

This positive obligation policy requires approval by NOC-Aadmin before any agents not specifically requested migrate into the NOC_A group:

Before Luna_NOC_A_Shared performs move agent where:
destination in group NOC_A
and not requested by 'NOC-AAdmin'
That Luna is obligated to obtain approval from 'NOC-AAdmin'

Obligation Policy Examples Combining Luna Agents and Environments. The Actors in an obligation policy may be a mix of Luna environments and agents. In this way, a Luna environment can respond to specified agent actions and vice versa.

For example, this positive obligation policy requires the Luna_NOC_A_Analysis environment to send a progress update every time a Botnet agent identifies a new botnet command-and-control address:

After BotnetAgent performs FoundC2
Luna_NOC_A_Analysis is obligated to perform SendAgentProgressUpdate

This positive obligation policy requires a class of agents that keep large data caches in RAM to clear their caches before being paused:

Before Luna performs PauseAgent where
Agent is of type CacheAgent
That Agent is obligated to perform DumpCache

7 Conclusions

This article has provided an overview of some of the unique features of the Luna agent framework. In particular, we have shown how Luna is specifically designed to allow developers and users to leverage different forms of policy-based governance in an endless variety of ways. Although our illustrations have been drawn from the application of Luna to cyber operations, we believe that its features will prove useful in the future for many additional problem domains.

8 Bibliography

1. Hoffman, R.R., J.M. Bradshaw, and K.M. Ford. "Introduction." In *Essays in Human-Centered Computing, 2001-2011,* edited by R.R. Hoffman. New York City, NY: IEEE Press, 2012.
2. Ford, K.M., C. Glymour, and P. Hayes. "Cognitive prostheses." *AI Magazine 18*, no. 3 (Fall 1997): 104.
3. Ford, K.M., and P. Hayes. "On computational wings: Rethinking the goals of Artificial Intelligence." *Scientific American. Special issue on "Exploring Intelligence" 9*, no. 4 (Winter 1998): 78-83.
4. Carroll, J.M., W.A. Kellogg, and M.B. Rosson. "The task-artifact cycle." In *Designing Interaction: Psychology at the Human-Computer Interface,* edited by J.M. Carroll. New York: Cambridge University Press, 1991.
5. Bradshaw J.M., M. Carvalho, L. Bunch, T. Eskridge, P. Feltovich, C. Forsythe, R.R. Hoffman, M. Johnson, D. Kidwell, and D.D. Woods. "Coactive emergence as a sense-

making strategy for cyber operations." *ICST (Institute for Computer Science, Social Informatics, and Telecommunications Engineering) Transactions of Security and Safety. Special section on "The Cognitive Science of Cyber Defense Analysis,"* 2012, in press.

6. Carvalho, M., T.B. Cowin, and N. Suri. "MAST: A mobile agent based security tool." *Proceedings of the Seventh World Multiconference on Systemics, Cybernetics, and Informatics* (SCI 2003), 2003.
7. Bradshaw, J.M., S. Dutfield, P. Benoit, and J. D. Woolley. "KAoS: Toward an industrial-strength generic agent architecture." In *Software Agents,* edited by J. M. Bradshaw, 375-418. Cambridge, MA: AAAI Press/The MIT Press, 1997.
8. Suri, N., J. M. Bradshaw, M. R. Breedy, P. T. Groth, G. A. Hill, R. Jeffers, T. R. Mitrovich, B. R. Pouliot, and D. S. Smith. "NOMADS: Toward an environment for strong and safe agent mobility." *Proceedings of Autonomous Agents 2000*, Barcelona, Spain 2000.
9. Bradshaw, J.M., P. Beautement, M.R. Breedy, L. Bunch, S.V. Drakunov, P.J. Feltovich, R.R. Hoffman, R. Jeffers, M. Johnson, S. Kulkarni, J. Lott, A. Raj, N. Suri, and A. Uszok. "Making agents acceptable to people." In *Intelligent Technologies for Information Analysis: Advances in Agents, Data Mining, and Statistical Learning,* edited by N. Zhong and J. Liu, 361-400. Berlin: Springer Verlag, 2004.
10. Klein, G., D.D. Woods, J.M. Bradshaw, R.R. Hoffman, and P. Feltovich. "Ten challenges for making automation a "team player" in joint human-agent activity." *IEEE Intelligent Systems 19*, no. 6 (November-December 2004): 91-95.
11. Bradshaw, J. M., P. Feltovich, and M. Johnson. "Human-Agent Interaction." In *Handbook of Human-Machine Interaction,* edited by Guy Boy, 283-302. Ashgate, 2011.
12. Uszok, A., Bradshaw, J.M., Lott, J., Johnson, M., Breedy, M., Vignati, M., Whittaker, K., Jakubowski, K., and Bowcock, J. Toward a flexible ontology-based approach for network operations using the KAoS framework. *Proceedings of MILCOM* 2011. New York City, NY: IEEE Press, November 2011, pp. 1108-1114.
13. Westerinen, A., *et al*., Digital Policy Management Ontology Discussion. Presentation at the DPM Meeting, January 25, 2012.
14. Carvalho, M., A. Granados, C. Perez, M. Arguedas, R. Winkler, J. Kovach, S. Choy. A Cross-Layer Communications Susbtrate for Tactical Environments. P. McDermott, L. Allender (eds.), Chap. 5, *Collaborative Technologies Alliance, Advanced Decisions Architecture*, 2009.
15. Bradshaw, J.M., M. Carvalho, L. Bunch, T. Eskridge, P. Feltovich, M. Johnson, and D. Kidwell. "Sol: An Agent-Based Framework for Cyber Situation Awareness." *Künstliche Intelligenz*: Volume 26, Issue 2 (2012), pp. 127-140.
16. EsperTech. http://esper.codehaus.org/ (accessed 18 July 2012).
17. Klein, G., P.J. Feltovich, J.M. Bradshaw, and D.D. Woods. "Common ground and coordination in joint activity." In *Organizational Simulation,* edited by W.B. Rouse and K.R. Boff, 139-84. New York City, NY: John Wiley, 2004.
18. Clark, H. H. *Using Language*. Cambridge, UK: Cambridge University Press, 1996.
19. Feltovich, P., J.M. Bradshaw, W.J. Clancey, M. Johnson, and L. Bunch. Progress appraisal as a challenging element of coordination in human and machine joint activity. In *Engineering Societies in the Agents' World VIII*, edited by A. Artikis, G. O'Hare, K. Stathis, and G. Vouros, 124-141. Lecture Notes in Artificial Intelligence 4995. Heidelberg Germany: Springer, 2008.