Dagger: Modeling and Visualization for Mission Impact Situation Awareness

Elisha Peterson
Johns Hopkins University Applied Physics Laboratory
11100 Johns Hopkins Rd, Laurel, MD 20723
443-778-2151
elisha.peterson@jhuapl.edu

***Abstract*—Dagger is a modeling and visualization framework that addresses the challenge of representing knowledge and information for decision-makers, enabling them to better comprehend the operational context of network security data. It allows users to answer critical questions such as "Given that I care about mission X, is there any reason I should be worried about what is going on in cyberspace?" or "If this system fails, will I still be able to accomplish my mission?"**

Keywords-Cyber situation awareness; network security; decision-making; modeling; visualization; mission impact assessment.

I. INTRODUCTION

Situation awareness always exists in the context of a larger goal. For decision-makers, that goal is expressed in terms of a *mission*, which might be characterized by the need to accomplish one or more specific tasks, or by the need to maintain the ability to accomplish these tasks. In network security, monitoring tools in common use today provide operators with knowledge of availability and vulnerability issues related to systems and applications, but often cannot translate that information into mission impact. This lack of knowledge affects all levels of an organization. The system operators do not understand how potential system changes, either for maintenance or to respond to an incident, will impact a mission. Similarly, decision-makers or mission performers do not know when or how their mission may be compromised due to lack of availability or security issues. The complexity of cyber exacerbates these issues, making it impossible for any one person to understand all of the relationships within the technology infrastructure, let alone the impact of changes on any particular mission.

Several efforts have been made to characterize the relationship between mission and cyber assets [1] [4] [5] [7] [10]. These works draw from related work in situation awareness and knowledge theory, such as Rasmussenn's 1985 paper on hierarchical knowledge representation [11], where the author identifies two challenges that are also central to mapping a mission to its cyber assets: gathering the appropriate information to represent this linkage and structuring that information appropriately. Visualization can help to present the information, but most recent work in network security visualization focuses on specific data sources rather than mission impact [6] [9] [12].

Dagger is a modeling and visualization framework that addresses the challenge of representing knowledge and information for decision-makers, enabling them to better comprehend the operational context of network security data. The modeling framework includes an iterative process for creating, improving, and maintaining a dependency model. Each component of a Dagger model supports user awareness in some way, either directly or by providing an abstraction to help map system state to mission awareness. The iterative modeling process is crucial for refining models as the system changes and for delivering initial value quickly while enhancing realism over time. Dagger includes multiple visualizations that help the user understand the operational context of cyber assets without overwhelming them. A layer visualization segregates assets and capabilities into multiple layers of abstraction, and a hierarchy visualization displays critical dependency paths between a focus item and other items in the system. Rather than displaying all dependencies all the time, the layer visualization displays dependencies on demand. Additional details related to a chosen item are also available on demand.

The remainder of Section I includes some additional background and discussion of related work. Sections II and III describe the components of a Dagger model and the modeling process. Finally, section IV describes Dagger's visualizations, emphasizing how they support situation awareness.

A. Related Work

Endsley defines situation awareness with three phases: (1) perception of objects or events in the environment, (2) comprehension of what those perceptions mean in the context of the current goal, and (3) projection of how those objects or events might change in the near-future [2]. Successful *mission impact situation awareness* involves aiding operators in perception of systems and applications, comprehending how system issues impact mission performance, and supporting predictions of how system and application changes will impact mission performance. Decision-makers have a representation of knowledge that links perceived information to their mission,

978-1-5090-3781-0/16/$31.00 ©2016 IEEE

and situation awareness systems for decision-makers should include a model to support that linkage. Rasmussen proposed representing such models as an abstraction hierarchy consisting of functional purpose, abstract function, generalized function, physical function, and physical form [11]. Flach, Mulder, and Paasen emphasized the importance of showing linkages within this hierarchy, stating *"in designing systems with the goal of good [situation awareness], it is the designers' responsibility to make the mapping across levels of abstraction visible to the operators"* [3].

In network security, there has been substantial work on modeling the mission impact of cyber-attacks [5] [7] [10]. In each case, the model contains elements of physical systems and the functions they support, as well as a subjective weighting of the importance of the physical systems or functions on the mission. The term "key cyber terrain" is sometimes used to describe physical or cyber assets in the context of mission requirements. Gathering the information required to build these models remains a challenge. The Camus system goes beyond previous work in partially automating the building of mission-dependency models, represented using an ontology, as well as in visualizing the mission impact [1] [4].

Dagger is designed to meet the situation awareness needs of decision-makers by showing mission-relevant information across multiple layers of abstraction, showing the linkages between these layers, integrating real-time data feeds, and showing additional information as necessary. Dagger is distinguished from prior work by the flexibility of its modeling framework, the ease of model development, the emphasis on user comprehension, and the choice of visualization (particularly the layer visualization).

II. THE DAGGER MODELING FRAMEWORK

A Dagger model describes the relationships within a collection of physical, informational, or abstract entities that make up or support a chosen system. The model includes *structural* components to describe the entities and relationships explicitly, and *computational* components to estimate the status of entities in the system (Figure 1). Structural components represent the "known state" of the system using the best available sensor data, while computational components represent an "inferred state" based on a predictive model.

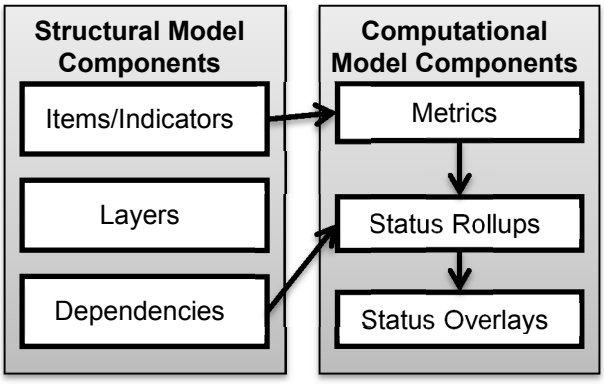

Figure 1. The components of a Dagger model include structural and computational components. Arrows show information flow. Metrics are computed using indicator information, and rollups use metrics and dependencies. Overlays consolidate status rollups for items in the model.

A. Structural Model Components

The structural components of a Dagger model are (i) a collection of related *items* and associated *indicators* that may change over time, (ii) a partitioning of items into an ordered set of conceptual *layers*, and (iii) a collection of *dependencies* between items that form a directed acyclic graph. An example of a Dagger model is provided in Figure 2.

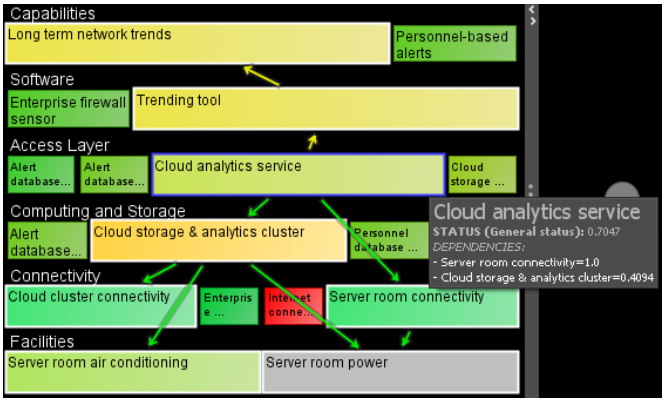

Figure 2. An example Dagger Model. Rectangles represent items, with indicators appearing when a mouse rollover is performed as shown in the grey popup box. Items are organized into layers labeled in white, and arrows show some of the dependencies between the items.

The *items* in a Dagger model constitute a collection of related and interdependent components and concepts. Possible items include (a) concrete and physical items such as network switches, servers, or power supplies; (b) software such as applications or enterprise services; (c) information such as a critical piece of data or the output of analytic; (d) specific people, groups, or organizations; and (e) capabilities, activities, objectives, or missions. Items may have observed values called *indicators*, such as the IP address of a computer, the temperature of a server rack, or the load on a CPU. Indicators might be manually or automatically associated with an item. They may also be either fixed or constantly changing.

Assigning items to *layers* of abstraction gives the user a way to quickly find items and understand their context. Missions may be grouped together on one layer, software on another layer, and physical systems on a third layer. Layers are ordered, with lower-level layers usually representing more concrete items such as physical systems, and higher-level layers usually representing conceptual items such as objectives and missions. The choice of layers needs to always be made with the end user in mind. From the cognitive engineering perspective, these layers are similar to those in an abstraction hierarchy in work domain analysis [11]. The abstraction hierarchy is traditionally defined with just 3-5 layers, but with a similar purpose of understanding the relationship between the functional purpose of a system and its physical components.

Several standard layer models that can be used in a Dagger model. The Open Systems Interconnection (OSI) model divides network entities into seven layers. The Joint Architecture Reference Model (JARM) Ten Layer Model, shown in Figure 3, extends the OSI model with the addition of people, capabilities, and missions, extensions that are critical for Dagger's use by decision-makers [8].

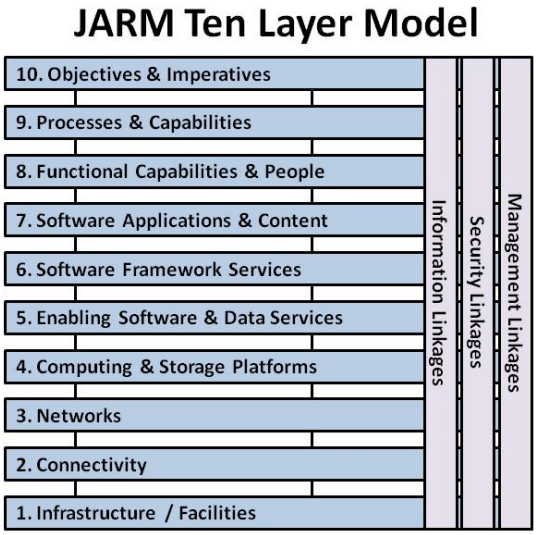

Figure 3. The JARM Ten Layer Model provides a way to frame many layers of abstraction within a single context (reproduced from [8]).

Dependencies in Dagger are simple statements such as "item X depends on item Y", and do not make any specific assumptions about the nature of the dependency. Simple dependencies are useful as a first-order model of the impact of component loss and/or degradation, and are also easily understood or identified by users. More complex dependencies can be included as part of the computational model. To ensure the computation of status is computationally tractable, the collection of dependencies is assumed to be *acyclic*.

B. Computational Model Components

The computational components of a Dagger model are (i) *statuses* of items, which have values between 0 and 1, or may be "unknown", (ii) *status rollups* of items, functions that operate on an item's indicators and dependencies to compute a status value, and (iii) *status overlays* of a system, representing a choice of status rollup for each item in the model. These components use indicators to compute an item's status on a fixed scale that provides a uniform way to assess value for all items in the model, such as the likelihood a certain indicator will degrade system availability. Multiple overlays can be used to represent different kinds of status information.

An *item status*, either a value between 0 and 1 or "unknown", generally represents an interpretation of observations associated with an item. For instance, the CPU load on a server might map to a value between 0 and 1 representing the likelihood that CPU overload might impact other services. Other indicators with statuses might include disk usage, RAM usage, temperature, etc. Generally, statuses will change with time.

A *status rollup* is a function that describes how to compute status for an item in the model. These functions depend both on the status of an item's dependencies and on the indicators of the item itself, and output a *status value* (i.e. between 0 and 1 or "unknown"). A *status overlay* is a choice of rollup function for each item in the system. For example, there might be one overlay representing availability and another representing security posture.

One difficulty with modeling status is that the term could mean many different things depending on context. For network *availability* monitoring, status might be the likelihood that a given component will be able to provide its intended functionality. This might be easy to measure for power systems or web sites, harder to measure for software, and impossible to directly measure for other entities such as people. At the abstract level, an appropriate status could be the *likelihood of being able to exercise that capability given the current state of the system*. For example, if an email server goes down, its availability is 0, whereas a user's ability to communicate a critical piece of information might be only partially degraded due to redundant paths such as phones.

The modeling framework addresses the inherent ambiguity in defining status by (i) providing heuristic status rollups that give a first-order understanding of system dependencies, and (ii) allowing status rollups to be customized for greater precision. The first heuristic status rollup is the *general status computation*. Given an item i, let $\{m_i\}$ be the set of the item's statuses, $\{m'_i\}$ be the subset of statuses with known values, $D(i)$ be the set of items i depends on, and $D'(i)$ the subset with known statuses. The general status is defined as:

$$s_g(i) = avg(\{m'_i\}) * avg\left(\left\{s_g(d) \middle| d \in D'(i)\right\}\right). \quad (1)$$

If an item has no known statuses, and none of its dependencies have a computable status, $s_g(i)$ is unknown. The second heuristic status rollup is the *level of awareness computation*:

$$s_a(i) = \frac{\|\{m_i\}\| + \sum_{d \in D'(i)} s_a(d)}{\|\{m_i\}\| + \|D(i)\|} \quad (2)$$

where $\|\cdot\|$ indicates the size of a set. Note that statuses with unknown values contribute to the denominator only. If an item has no statuses or dependencies, its level of awareness is set to 0. In contrast to general status, the level of awareness always produces a value between 0 and 1.

The general status computation is good for first-order modeling in a variety of scenarios. If the statuses are interpretations of system availability, it provides an estimate of likely mission/capability availability. The level of awareness computation is good for almost any scenario, assuming only that the model includes statuses of required known attributes with null values when they are missing. Other variations on these formulae may be more natural for certain models. For instance, a *maximum* value might be used in place of an average if the dependencies tend to be redundant, or a *minimum* might be used if the dependencies are all required. An argument could be made that unknown dependencies should be factored into the status computation as well. Ultimately, the choice of a specific heuristic depends on the system to be modeled, and will almost always need to be further customized.

III. THE DAGGER MODELING PROCESS

Constructing a Dagger model for a chosen system is an iterative process that involves four steps: (i) identify the *system* to be modeled and the scope of situational awareness desired, (ii) identify the *system structure*: the items, the layered structure of those items, and the dependencies within the items,

(iii) identify available *data feeds* that can be mapped to items in the model; determine appropriate mappings for the status of interest, and (iv) *iterate* to build more realism into the model and/or adjust the model to match system changes.

Two considerations drive the iterative approach. First, it takes a substantial effort to identify the true scope of a system, gather the appropriate subject matter expert (SME) input to construct the model and dependencies, and integrate the data feeds. Second, in many cases the system routinely changes, as items are added or removed. It often makes sense to build a first model quickly to provide a basic feel or starting point for the system to be modeled. These low-fidelity models are useful not only as an initial capability for the end-user, but also as a starting point to engage SME's and gather necessary information for more precise models. With further analysis and expert feedback, more complete models may be developed. It also makes sense to iteratively gather status feeds. Generally not all items in a model will be instrumented, and the initial model may be used to help prioritize additional instrumentation (e.g., if it is more important to have confidence in the status of one mission over another). In any given application, one must balance the need for fidelity against the value a simpler model might provide.

A. Modeling the System

Systems can be modeled from the top-down, starting with key missions or capabilities, from the bottom-up, starting with physical infrastructure, or using a hybrid approach. In any approach, care must be taken to scope the model so that only relevant items are included (otherwise, the model would quickly become unmanageable). Developing the model requires inputs from multiple subject matter experts with different specialties. One SME might provide the critical missions and mission essential tasks but have no knowledge of the components and systems that support those missions, while another SME might know the infrastructure that supports those missions but not much about the missions themselves. The resulting structural model often includes 5-15 layers, several items within each layer, and dependencies between the items.

A challenge that often occurs in top-down modeling is the exponential growth of dependencies. If a particular subsystem in the model has k subsystems, each of which has another k subsystems, etc., the model quickly becomes too large. The modeler must choose a level of abstraction of status for each of the subsystems that is appropriate for the end user's decision-making needs, keeping in mind that much of the detail of the model will be hidden. The modeler also needs to understand what kind of data is available to feed the model.

B. Modeling Dependencies and Status

A general dependency indicates that one item is dependent upon another item or items to operate as expected. Dependencies sometimes exist in existing databases for elements in concrete layers, but usually require SME input as well, particularly for the more conceptual layers.

Identifying dependencies is intertwined with modeling of indicators and statuses. The general status heuristic can be used to illustrate linkages between items. If this is insufficient, the modeler should identify key nodes where the general status rollup is not adequate, for instance "OR" dependencies, where an item requires one or more (rather than all) of several other items to be operational. Because Dagger uses one status overlay for both conceptual layers like "mission" and concrete layers like "servers", regardless of how quantitatively dependency modeling is done at the lower layers, presentation of status at the higher, abstract layers will almost always be the result of more qualitative considerations.

Another issue to keep in mind while modeling status is the availability of data to support the status. For instance, in networked systems, the concept of "availability" is much easier to obtain and measure than "confidentiality." This is one of the reasons models include the concept of *unknown* or *incomputable* statuses.

C. Incorporating Data Feeds

In real-time situation awareness, item indicators and statuses may arise in Dagger in three different ways. First, they may be a part of the model. Second, a manual entry process may be used. Third, they may result from real-time data feeds. In practice, many models will use all three processes. The model may include static indicators of key items that typically don't change. Manual input can be useful for items whose status is not measured, such as when they are not instrumented or when human judgment is required to assign status.

IV. VISUALIZATION OF DAGGER MODELS

Dagger's visualization techniques are designed to present item status in a way that is useful for decision makers to understand the impact on the mission. Two key questions they address are *"What is the status of mission/capability X?"* and *"Given that mission X appears to be degraded, what is the problem, and what are the next steps to fix it?"* Rather than showing as much information as possible at one time, the visualizations rely on effective use of information hiding, following the "overview first, zoom and filter, details-on-demand" mantra championed by Schneiderman [13].

A. Layer Dependency Visualization

The *layer visualization* groups model items into conceptual layers, which are organized from lower-level items (usually physical systems) to higher-level items (usually missions and capabilities) according to the model. The layering helps the user quickly find items of interest. Within each layer, items are presented as colored, labeled rectangles. Status is presented by coloring items: by default red indicates 0, yellow indicates 0.5, and green indicates 1. Interpolated colors are used for intermediate values and gray indicates "unknown". Alternate color schemes can be configured for color blind users.

Figure 4 shows a hypothetical model with six layers, representing a simplified system that senses and alerts on network traffic. The statuses are computed using the heuristic general status rollup defined earlier, with a few indicators providing the raw data. This system provides two high-level capabilities: an understanding of long-term trends, and personnel-based alerts that fire when an employee's computer accesses the Internet but they are not in the building. A hypothetical "personnel database" contains this locational information. Long-term trending is supported by a cloud

storage and analytics cluster. Using the visualization, the user would quickly notice problems with Internet connectivity and the cloud cluster. They might also notice that there is no information on server room power, and that the long term network trend capability is degraded. Their response to these issues would depend on the context (e.g., is this a common or uncommon state of the system? Is Internet connectivity necessary?)

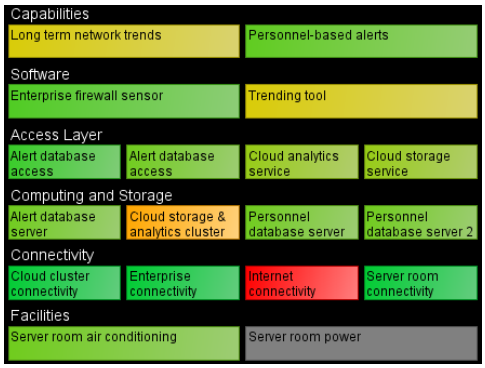

Figure 4. Layer model for a network alerting and trending system (simplified), showing the general status overlay.

Dependencies are not displayed in the default layer visualization, since there are often too many to display effectively. Instead, when the user selects or moves a mouse over a particular item, all other entities in its *dependency chain* are highlighted. In Figure 5, one sees everything that the "cloud analytics service" supports, and everything it requires to execute. Together with the tooltip, one can see that degradation to the service is caused by degradation of the cluster.

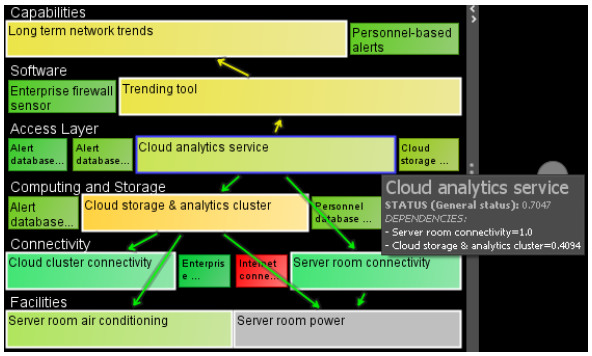

Figure 5. The layer visualization highlights dependencies when the user selects or moves the mouse over an item. Arrows indicate dependencies, where green arrows indicate "depends on" relationships, and yellow arrows indicate "supports" relationships (the inverse of "depends on").

The "level of awareness" overlay can be selected to provide an alternate view of the system, shown in Figure 6. In this view, red indicates a complete lack of awareness, yellow a partial awareness, and green a completely adequate awareness of an item. This shows how lack of knowledge of server room power might impact a confident assessment of system status.

To support navigation of large models, Dagger supports two kinds of filtering. Filtering by layer helps mission-level decision-makers see their mission-readiness without unnecessary detail. Filtering by item allows the user to focus on just the part of the system related to the given item.

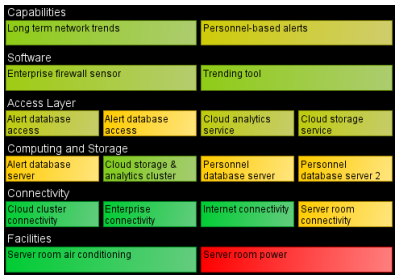

Figure 6. The level of awareness overlay shows how lack of information about one item might influence one's confidence about status.

B. *Other Views*

Clicking on an item populates a view with additional details, as in Figure 7. This view shows that the cloud cluster has an unusual "average load," which might degrade response times. The view can also show pertinent information relevant to a decision-maker, such as the point of contact for the problem or a URL link to another tool.

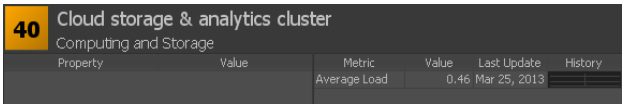

Figure 7. Details of a selected entity.

The *sunburst* visualization displays each unique dependency path explicitly by mapping the directed acyclic graph of dependencies into a tree. Dependencies are either directed outward (Figure 8 left) or inward (Figure 8 right), and the same entity might be shown multiple times. This visualization is good for quickly identifying what is causing the degradation of an item at interest. This view is shown whenever the user clicks an item in the layer visualization. Together with the item details panel (Figure 7), the sunburst view helps the user quickly understand a situation.

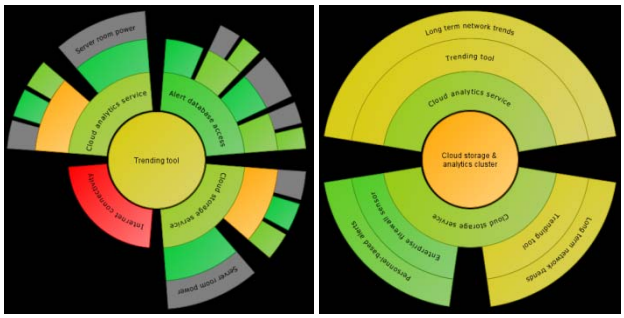

Figure 8. The sunburst visualization shows explicit dependency paths. At left, one sees what the "trending tool" depends on, and at right what items the "cloud storage & analytics cluster" supports.

C. *Predictive Analysis*

The status overlay provides a mechanism for basic predictive analysis of a system, addressing questions like *"What happens to mission X if entity Y is degraded?"* This question can be addressed by assuming a fixed value of a chosen item's status, or a set of chosen items, with the remaining status rollups computed normally. Figure 9 shows status predictions when the server room has no power, and when it has full power. One would expect in a realistic model that servers would be completely unavailable when the power

is down; the presented status is computed via the general status heuristic, and could be customized for greater realism.

Figure 9. "What-if" analysis provides a predictive analysis capability. The two figures show the predicted status of the system when "server room power" is unavailable and when it has full power.

V. CONCLUSION

Dagger includes a modeling framework with structural and computational components, a visualization framework, and a process for integrating real-time data feeds and refining the model to accommodate change or greater realism. By hiding details that need be presented only when the user demands them, the visualization makes it easy to quickly find and assess the status of items of interest. The dependency model enables calculation of status for abstract items such as missions, capabilities, and mission essential tasks, supporting high-level mission impact situation awareness. The framework is flexible and has been applied to information systems, research systems, personnel management, task management, space systems, and hybrid cyber-physical systems. In each case, the first iteration of the model was created in a few days or weeks. The initial model provided awareness to users, while also helping modelers identify key instrumentation requirements and areas to improve the model.

A. Future Work

A key challenge is building *scalable* Dagger models. If scaled to thousands or millions of items, the bottleneck would be identifying items and dependencies. (The computation of status scales linearly, given the directed acyclic graph constraint on dependencies.) Because of this bottleneck, additional work on automated network dependency discovery would contribute to greater scalability of Dagger models.

An alternate way to scale models is to integrate multiple related Dagger models at different levels of abstraction. A system might have many subsystems represented as a single item in its Dagger model, but these subsystems might have their own Dagger models. In this scenario, a family of Dagger models would form a hierarchy where the lower tier models feed the higher tier models, and the status of a Dagger model becomes an output that is made consumable as a service. This hierarchy of models might be built by a corresponding hierarchy of organizational units.

Many real world systems, particularly critical infrastructure such as power systems, include cyclic dependencies that Dagger's modeling framework does not directly support. Future work may identify ways to either model such systems within Dagger's current framework, or extend the modeling framework to support cycles.

Modeling the state or change of the system over time is another open challenge. The dependencies in many systems change with time, or might be state dependent, whereas a Dagger model as described here includes fixed dependencies. One possible solution is to model different system states using different overlays. The models could also be extended to incorporate predictive models of the system. For instance, the predicted reboot time of a server might be leveraged to show the time until a particular system is again operative.

ACKNOWLEDGMENTS

Initial work designing the layer visualization and associated model was completed by a team including John Forte, Jeff Osborn, Marcos Osorno, and Tom Spriesterbach. The author is also grateful to Jeff Chavis, Brian Garofalo, Jennifer Ockerman, Pedro Rodriguez, Virginia Walker, Brenda Wetzel, and Keith Wichmann for helpful feedback.

REFERENCES

[1] Buchanan, L., Larkin, M., & D'Amico, A. 2012. Mission assurance proof-of-concept: Mapping dependencies among cyber assets, missions, and users. *Homeland Security Technology,* 298–304.

[2] Endsley, M. R. 1995. Toward a theory of situation awareness in dynamic systems. *Human Factors, 37*(1), 32-64.

[3] Flach, J., Mulder, M., & Paassen, M. Van. 1995. The Concept of the Situation in Psychology. *A cognitive approach to situation awareness: Theory and application*, 42-60.

[4] Goodall, J. R., D'Amico, A., & Kopylec, J. K. 2009. Camus: Automatically mapping Cyber Assets to Missions and Users. *Proceedings of the 2009 IEEE Military Communications Conference*, 1–7.

[5] Grimaila, M. R., Mills, R. F., & Fortson, L. W. 2008. Improving the cyber incident mission impact assessment (CIMIA) process. In *Proceedings of the 4th annual workshop on Cyber security and information intelligence*.

[6] Harrison, L., & Lu, A. 2012. The future of security visualization: Lessons from network visualization. *Network, IEEE*, 6–11.

[7] Jakobson, G. 2011. Mission cyber security situation assessment using impact dependency graphs. *Proceedings of the 14th International Conference on Information Fusion (FUSION).*

[8] Marks, R. 2011. Enterprise Architecture in the Intelligence Community. *6th Annual DOD Enterprise Architecture Conference*.

[9] Marty, R. 2008. *Applied Security Visualization*. *Network Security*. Addison-Wesley Professional.

[10] Musman, S., Temin, A., & Tanner, M. 2010. Evaluating the impact of cyber attacks on missions. *Proceedings of the 5th International Conference on Information Warfare and Security*, 1–15.

[11] Rasmussen, J. 1985. The role of heirarchical knowledge representation in decision making and system management. *IEEE Transactions on Systems, Man and Cybernetics*, *15*, 234-243.

[12] Shiravi, H., Shiravi, A., & Ghorbani, A. 2011. A Survey of Visualization Systems for Network Security. *IEEE Transactions on Visualization and Computer Graphics*.

[13] Shneiderman, B. 1996. The eyes have it: a task by data type taxonomy for information visualizations. *Proceedings 1996 IEEE Symposium on Visual Languages*, 336–343.