1.1 Anti-Terrorism Planning and Resource Allocation

A substantial stream of ATP systems research has emerged that addresses issues related to the challenges of supporting intelligence gathering and analysis, and emergency and crisis response [1, 2]. Less research has so far been aimed at the more routine but arguably more prevalent activity of defensive ATP, which involves identifying and prioritizing what should be protected, and the most effective AT mitigations that can be applied within cost constraints. This activity is carried out regularly at all levels of government from national to local, and within large corporations. Supporting this emerging population of non-expert users with sophisticated ATP systems presents special challenges for decision support system researchers and designers.

Much of the reported research in the AT domain has focused on the design and development of information retrieval and aggregation tools for analysis of large data sets [3], tools for modeling terrorist attacks [4], collaborative tools for counter-terrorism [5], cyber security [6], and managing privacy [7]. Significantly less evident in the literature are studies describing the design and evaluation of systems designed to support anti-terrorism activity, such as planning, decision-making, and emergency response, and what can be done to make these systems more effective and efficient.

Additionally, there is limited reported research on development of agent environments for real-world complex environments. One of the few research reports on agent environments for complex problems has been the Phoenix forest fire simulation environment [8]. We believe that our work also contributes towards the research knowledge in this domain.

In this paper, we describe an approach to service-oriented and agent-centered architecting of complex, web service-based applications and systems. Our objective is to develop a design reference model for ATP that views such systems as consisting of a network of human and technological actors working together to perform complex cognitive tasks. We describe the application and demonstrate the potential utility of these tools through a reference design of a complex, web service-based system to support anti-terrorism planning.

1.2 U.S. Marine Corps ATP

Anti-terrorism planning is a distributed, high-priority activity at Marine Corps installations. Personnel in a number of different roles are concerned with ATP including public works officers (typically civil engineers from the Navy), anti-terrorism officers, provost marshals (military police), command, and civilian facility planners. They identify and assess mission priorities, threats, and vulnerabilities; assess and select appropriate ATP mitigations; write ATP project plans and specifications; manage ATP budgets; and work with commanding officers and the headquarters Marine Corps to ensure that ATP objectives are being met. The types of projects they manage include ATP mitigations for new construction, ATP upgrades and retrofits to existing structures, and structures such as blast walls purpose-built to protect identified mission critical areas.

The study reported here involved work with the United States Marine Corps on development and evaluation of a decision aid for defensive anti-terrorism planning and resource allocation. The system is called Rampart, and implements a hybrid decision model for asset prioritization, determination of mitigation project utility, and optimal allocation of constrained resources. In addition, it includes tools to support user learning, improve user performance, and assist users in reflective analysis of computed results.

2 The Rampart ATP System

Our early work on ATP requirements identified three major problem sub-components in the ATP planning domain: prioritization of facilities and other assets to be protected, computing the relative utility of different AT mitigations and mitigation projects (the latter are combinations of mitigations), and allocation of available resources (money, time, people, or other resources) to protect the highest priority facilities with those mitigations providing the highest utility. A high-level view of this conceptualization is provided in Figure 1.

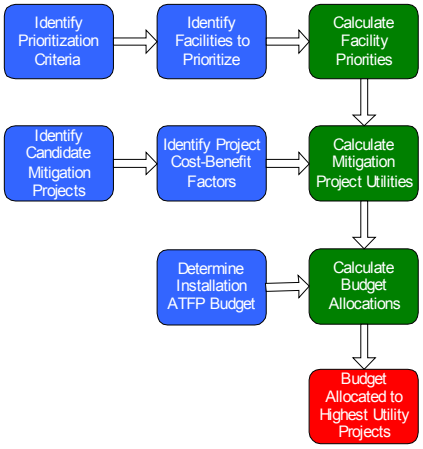

Fig. 1. The Rampart ATP Conceptual Model.

Anti-Terrorism Officers (ATO), facilities planners, or emergency response personnel (henceforth referred to as users) require the functionality shown in Figure 1 because they have assets (e.g., different facilities at a Marine Corps installation) to protect, as well as a budget to be allocated among some mitigation projects, which in turn are created to protect a given asset. Mitigations are those activities and materials that can provide protective benefit, such as installing glazing upgrades or placing stand-off barriers around mission critical facilities. Mitigations provide protective benefit by protecting the facilities, the equipment housed in them, and, most importantly, the people who work or live in them. The facilities are ranked according

to a set of criteria where their resulting weights represent the relative importance of each one.

User activities include the identification of prioritization criteria, pair-wise weighting of these criteria, identifying the different infrastructure facilities to be protected, pair-wise weighting of the facilities against the previously identified prioritization criteria, and design of mitigation projects that might be applied to identified facilities. For example, stand-off barriers (the mitigation project) paced around a tank farm (the facility) at an oil refinery (the infrastructure). The costs of the mitigations, the relative importance of the various facilities and the available budget are used as input for a mixed-integer linear program to arrive at an optimal resource allocation. Users also have the flexibility of adjusting the system-generated results by using the override features that are provided for each sub-task in the planning process.

2.1 A Service- and Agent-oriented Architecture for ATP

The vision of the semantic web [9] suggests a future computing environment in which humans, intelligent agents, and web services interact to carry out complex tasks. Design theories and principles for systems conforming to this vision are, however, fragmented into issues specific to the sub-fields of usability engineering for humans, software and knowledge engineering for intelligent agents (agent-oriented software engineering), and web service architectures. This presents a challenge for developers of these systems because they lack a source that integrates these disparate ideas into a cohesive design framework. The Rampart architecture is based on the model-view-controller (MVC) architectural pattern, as shown in Figure 2.

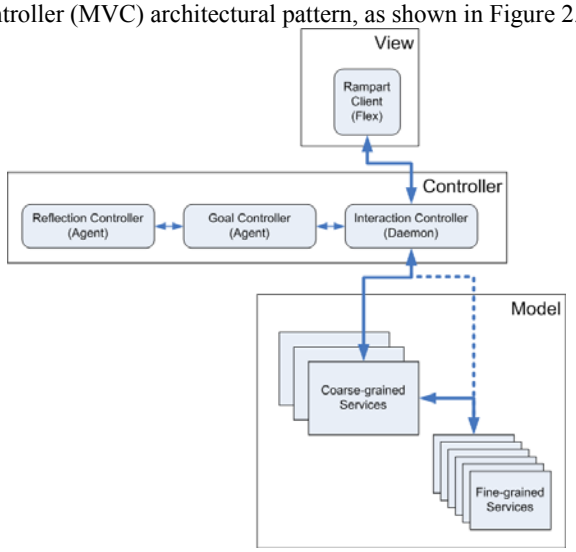

Fig. 2. The Rampart Agent Architecture

As shown in Figure 2, the model layer consists of a set of coarse and fine grained web services that provide the fundamental decision making functionality. Coarse-

grained services are those aligned with the goals of a Rampart end user. For example, *Add Prioritization Criteria* or *Compute Resource Allocation*. The finer grained services are specific smaller service components that contribute to achieving the coarse grained service functionality. The controller layer consists of a piece of custom middleware, the *Interaction Controller*, and two agents we call the *Goal Controller* and the *Reflection Controller*. The controller layer is instrumental in managing the interaction between the view (web pages), the specific web service requests, and agent suggestions, based on the state of the system. Besides directing the application workflow, the controller is also responsible for managing the exchange of messages between the view and model layers.

The view layer is the *Rampart Client* implemented as an Adobe Flex™ web application. This paper focuses on the first two of these three levels (i.e., the model and controller levels); the view, or web client layer will not be discussed further.

This implementation of the MVC architectural pattern separates the system into three levels of abstraction. The MVC facilitates the process of decoupling where functions can be created as independent modules with no knowledge of the other modules. In this case, only the controller has information about the modules either at the model level or at the view level. The MVC paradigm allows each level to be implemented independent of their physical location (i.e. server or other computer), which is an important feature and benefit of modularity.

2.2 Rampart Web Services (Model layer)

One of our goals in the development of Rampart was to use web services to encapsulate the fundamental units of decision making and other support services provided to the system's users. Initially, this was driven by the need to have flexible and re-composable web services. Requirements in the ATP domain change rapidly and we wanted the ability to 'plug' and 'unplug' services without making major changes to the web client. The use of web services as functional modules also contributes to the system's scalability and more efficient performance. As the demand for a particular system capability increases, the related web services providing it can be re-deployed to a different server to avoid potential overload.

Web services provide a single, discoverable interface standard for computing functionality. One of our objectives here is to leverage this homogeneity so that people and agents can discover, comprehend, and use services in a flexible manner that allows them to compose solutions to ATP problems. The use of web services to provide Rampart's core functionality also allows that functionality to be easily used in other, related applications, such as a dynamic simulation environment.

The design of Rampart has taken into consideration the discrete functionalities of the system and the division of these functionalities into modules. Seeking the most appropriate integration of the modules (cohesion) with the least amount of dependencies between them (coupling), we developed a flexible composition schema (see Figure 3). An ATP task analysis was used to predict how modules should be connected, and how modules should be defined to increase cohesion, on the one hand, and to decrease coupling, on the other. Each module represents a complete, discrete

functionality within the system, except for those high-level modules that are formed by the composition of other modules.

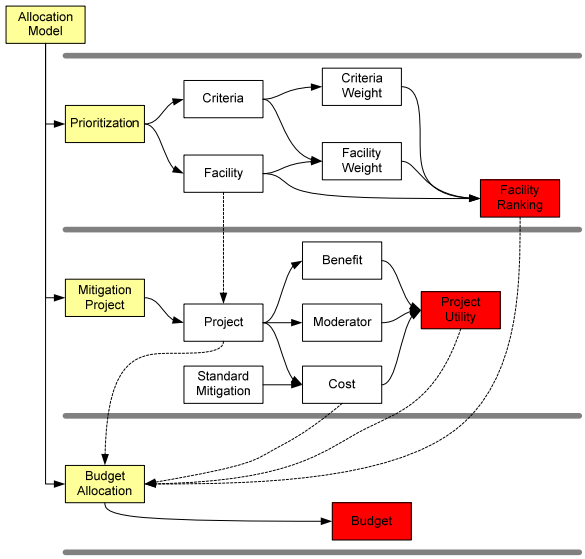

Fig. 3. Rampart Services Composition Schema

Figure 3 provides a simplified depiction of the dependencies between different Rampart services. The dotted lines represent the dependencies between modules, which are important representations of modular (service) cohesion and coupling. Although, the system presents a good level of modularity, the data structure and the service composition still poses a limitation for achieving the high cohesion and low coupling necessary for efficient service reuse and extensibility.

The service with the least cohesion is *Budget Allocation*, which is composed of capabilities related to the allocation model, cost-benefit factors, mitigation projects, facilities weight, and cost. This module also presents the highest level of coupling due to its data dependencies, so-called data coupling, where inputs for budget allocation come from the results of other modules. Figure 4 provides a further decomposition of the model (web service) layer showing the operations implemented in the criteria service.

The criteria service consists of three coarse-grained services representing the tasks the user must perform. As part of completing their resource allocation task, a Rampart user must create/edit criteria, perform a pair-wise weighting of the criteria, check for the consistency of the criteria weights, and view the relative ranking of the criteria based on the weighting. These tasks are represented using the three coarse-grained services: *Edit Criteria*, *Weight Criteria*, and *Show Criteria Scores*. Each of these coarse-grained services has several underlying fine-grained operations that support the higher level service. For example, the Edit Criteria service consists of several

operations including getting the list of current criteria, and inserting, updating, or deleting criteria.

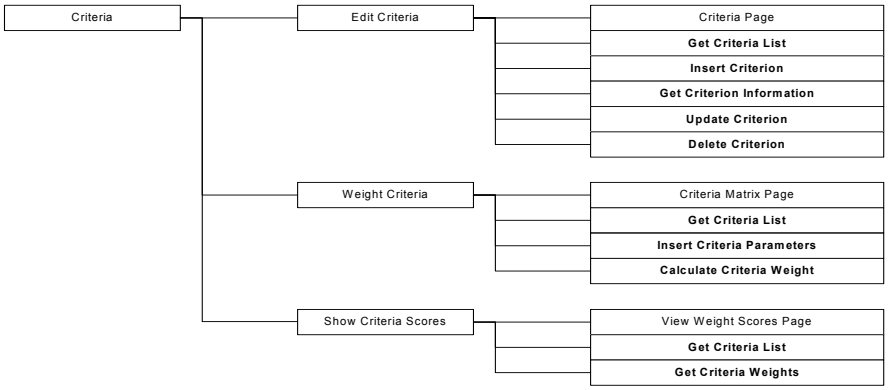

Fig 4. The Rampart Criteria Web Services

Despite the general opinion that the module and view can communicate directly to each other for the purpose of message passing, Rampart uses a controller based centralized architecture for message exchange (i.e., for receiving and forwarding). This approach reduces coupling between modules because each layer becomes an isolated and independent module and increases granularity and composition because each layer is then responsible for its own functionality. In other words, a function can be handled by only one of the layers.

Figure 5 shows an example of the message passing sequence in the Rampart environment. In the figure, the sequence of message passing between the web client, the interactive daemon controller, agents (goal and reflective), and the web service is shown. While this process involves several levels of message passing (between the agent, client and the services), the control solely rests with the daemon. This control is useful, especially in the goal controller agent, where decisions are made in response to user actions.

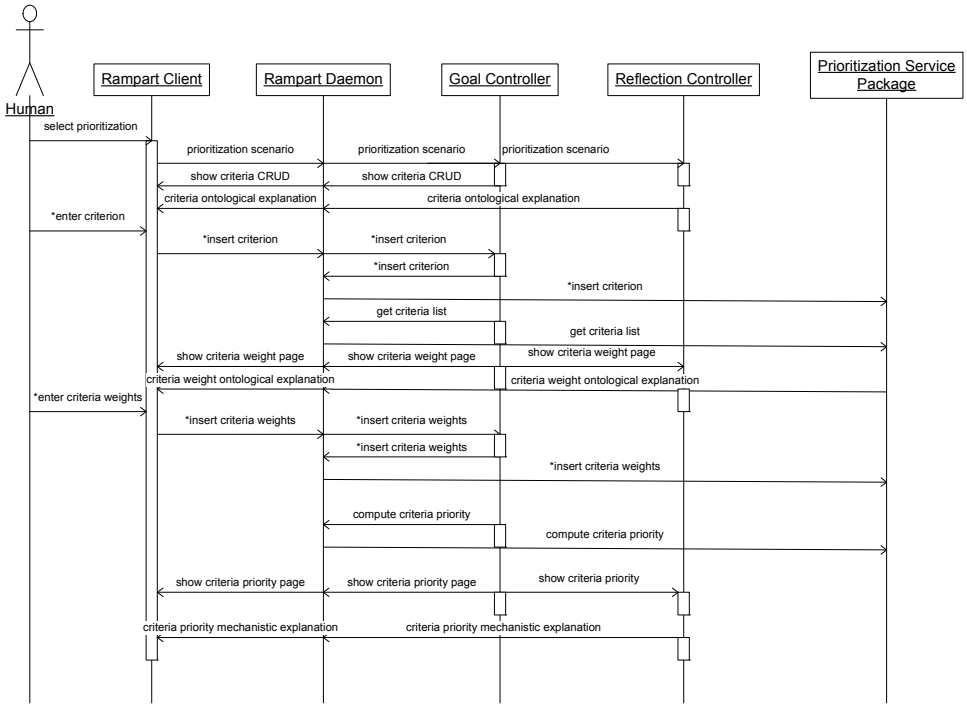

Fig. 5. The sequence of message passing in the Rampart environment

2.3 Rampart Agents and Daemon (Controller layer)

In the classical MVC architecture pattern, the controller layer mediates between the interactions of the user with the view layer (e.g., the Rampart web client) and the core functionalities provided in the model layer. The Rampart controller layer is composed of two Soar [10] agents and customized middleware (the Rampart *daemon*) designed to mediate between the Flex-based Rampart web client and the core Rampart web services. The daemon is responsible for message passing, acting as a sort of 'traffic cop' between requests from the user, the functionality embedded in the web services, and the supporting knowledge embedded in two intelligent agents.

The Rampart *goal controller* and *reflection controller* agents were developed using the Herbal agent development environment [11] and the *Soar* cognitive architecture [10]. The purpose of the goal controller is to help align the actions of the user, especially novice users, with a normative model of the ATP and resource allocation task. The goal controller knowledge base was derived from an empirical analysis of

Marine Corps anti-terrorism planners using an earlier version of the system [12]. This field evaluation of Rampart suggested that the system's diverse user base requires significant support to create effective ATP models.

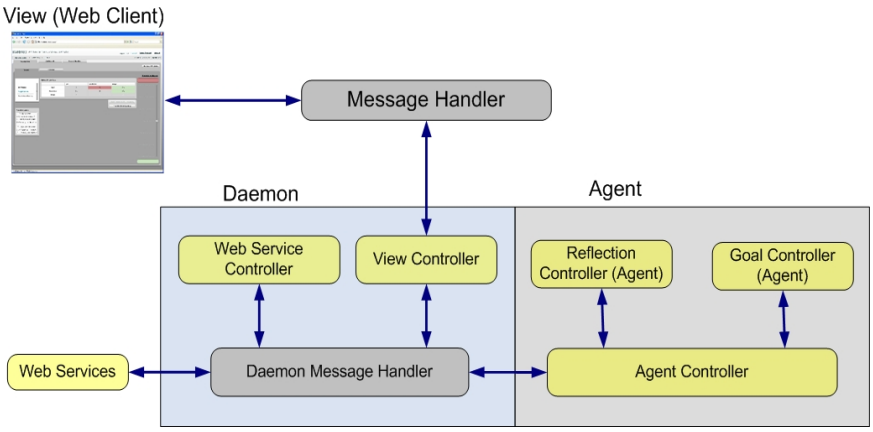

Fig. 6. Rampart daemon architecture

As described earlier, the daemon interacts with the web services and the client through a sequence of message handlers. As shown in Figure 6, the Rampart daemon acts as the middleware between the web client and the services. The daemon architecture handles messages coming from the user (web client), web services, and the agent. The sequence of actions can be described as follows: a user action (e.g., user clicks add new criteria) will trigger an event to check for possible suggestions from the agent. Based on the agent suggestion, web services are instantiated and the user is directed to the appropriate page to add the criterion. Any agent suggestion is also provided to the user at this time.

The purpose of the Rampart reflection controller agent is to act as a design critic [13], examining the user's evolving ATP model as it is constructed and offering advice to improve it. Model building in Rampart is a form of design problem solving. The solution to be designed is a planning model that identifies and weights the appropriate prioritization criteria, facilities, mitigation projects, and mitigation project utility factors. Like most design problems there are constraints, in particular, the major constraint is the budget available to support protecting a particular infrastructure. The reflection controller supports the user by its awareness of these constraints and their effect on the possible improvements to the planning model.

The reflection controller is designed to act as a human ATP expert looking over the user's shoulder. Essential to providing effective support is tracking the user's actions, knowledge of their goals and intentions, an understanding of the system's functionality, and awareness of the current state of the user's ATP model with respect to the form of an idealized or normative model.

Our work so far on the Rampart agents has focused on the relatively easy tasks of supporting ATP workflow and basic critiques of the ATP model as it is constructed.

Much more difficult is establishing an understanding of the user's intentions and goals from using the system. Moving forward we are interested in exploring more sophisticated ways of understanding the user's context and for supporting them more effectively in their work with the system.

3 Lessons Learned

Rampart's web service architecture has provided significant benefits during system re-design and extension. The relatively low coupling between the model (web service) and other MVC architecture layers allowed us to create an entirely new web-based client and to integrate intelligent agents into the systems workflow without major changes to the core functionality implemented in the web services themselves. Rampart's architecture also supports re-purposing the core decision support functionality in the model layer. For example, any one of the three packages of functionality (prioritization, project utility, or resource allocation) can be accessed as stand-alone modules to support a more limited-use client. At the same time, Rampart users could also decide to integrate third-party prioritization ranking, project utility, or allocation web services to meet specialized ATP requirements.

One of the major drivers of granularity in web services is performance. High-level, outward facing services can play a role as a cache for subsequent calls to finer grained services that are easier for developers to reuse and for agents to compose into customized services. This approach to web services architecture does, however, present performance challenges, and these may be particularly important in domains such as agent-based simulations operating in real time.

Service composition is one of the most important paradigms of web service based applications. In large complex systems, composed of multiple, interdependent sub-systems, it is important to design with the purpose of re-using the available components. Service composition also affords orchestrating new workflows with relative ease by re-organizing the services to fit the requirements of various user groups.

We have struggled with the question of whether the agent should provide assistance with a "one rule at a time" model versus a full model critique. In the first case, the goal controller agent provides suggestions on what to do next, based on the current state of the model while the reflection controller examines only the last part of the model completed. The latter case involves the agents interacting to examine the entire model in its current state and providing the user with more 'global' suggestions, in other words, everything required to bring the current model to completion.

The Rampart architecture and its supporting agents have so far focused on providing the user with advice based on application knowledge rather than on the broader class of ATP domain knowledge. Building more domain knowledge into the agents —knowledge of terrorism as an activity, civil and mechanical engineering, and the relative effectiveness of different anti-terrorism mitigations— would enhance the power of the system as a tool for supporting not only ATP tasks but also as a resource for learning about the ATP domain more broadly.

One of the ways we are approaching this problem is through the development of an integrated development environment (IDE) for agents and cognitive models capable of communicating their design rationale. The IDE, which we call Herbal [11], was designed from the ground up as a tool for creating agents that are made easier to develop, use, maintain, and evolve through explanation and design rationale, among other features. The IDE is in its second major version; it is operational and available for use by others interested in creating friendlier and more articulate intelligent technologies (See http://acs.ist.psu.edu/herbal/ for more information).

3.1 Moving Forward

Web services are by design only aware of their own functionality; it is up to humans, agents, or applications to compose multiple services into a coherent application. One of our goals moving forward is to allow the Rampart agents to direct service composition. As it stands currently, the agent is only aware of the services underlying Rampart in terms of the functionality they provide, but not of how each service and its operations contribute to a specific end user goal. We are actively researching ways to infer user goals, for example, through the use of a standard catalog of ATP scenarios to draw on as a form of case-based reasoning.

One of our goals is to simplify the Rampart daemon middleware into a more generic set of agent-based *explanation services*. These services would appear and behave much like our current web service interface and would provide a consistent way of using the functionality provided by the goal controller and reflection controller agents. This service package would also provide tools to allow system users and developers to gain access to details of the agents' operations. These services would be focused on componentizing the agents' explanation content so that both users and developers get what they want, when they want it.

4 Conclusion

In this paper, we describe an architecture design reference model for supporting users of complex, ATP systems. The system employs a web service architecture and incorporates intelligent agents to support application usability. The system was developed over several cycles of formative and summative evaluation with potential users. The architecture and its rationale are useful as a reference design model for both researchers and practitioners setting out on ATP and development projects.

Acknowledgments
This work was supported by the U.S. Office of Naval Research (ONR) under contract N00014-06-1-0164. The United States Marine Corps supported development of the Rampart ATP system. We would also like to thank Laxman Vembar and Maik Friedrich for their contributions to programming some of the components of the Rampart Agent.

References

1. Chen, H. *Intelligence and Security Informatics for International Security: Information Sharing and Data Mining*. Springer, 2006.
2. Yen, J., Popp, R., Cybenko, G., Taipale, K.A., Sweeney, L. and Rosenzweig, P. Homeland security. *IEEE Intelligent Systems*, *20*, 5, (2005).
3. Avant, D., Baum, M., Bertram, C., Fisher, M., Sheth, A. and Warke, Y., Semantic Technology Applications for Homeland Security. in *Proceedings of International Conference on Information and Knowledge Management (CIKM 2002)*, (McLean, VA, 2002), 611--613.
4. Popp, R. and Poindexter, J. Countering Terrorism through Information and Privacy Protection Technologies. *IEEE Security and Privacy*, *4*, 6, (2006), 18-27.
5. Popp, R., Krishna, P., Willett, P., Serfaty, D., Stacy, W., Carley, K., Allanach, J., Haiying, T. and Satnam, S., Collaborative Tools for Counter-Terrorism Analysis. in *Aerospace Conference, 2005 IEEE*, (2005), 1.
6. White, G.B. and DiCenso, D.J., Information Sharing Needs for National Security. in *System Sciences, 2005. HICSS '05. Proceedings of the 38th Annual Hawaii International Conference on*, (2005), 125c.
7. Davies, S. A year after 9/11: Where are we now? *Communications of the ACM*, *45*, 9, (2002), 35-39.
8. Cohen, P.R., Greenberg, M.L., Hart, D.M. and Howe, A.E. Trial by Fire: Understanding the Design Requirements for Agents in Complex Environments. pp. 32-48, 1989. *AI Magazine*, *10*, (1989), 32-48.
9. Berners-Lee, T., Hendler, J. and Lassila, O. The semantic web. *Scientific American*, *284*, 5, (2001), 34-43.
10. Laird, J., Newell, A. and Rosenbloom, P. Soar: An Architecture for General Intelligence. *Artificial Intelligence*, 33, (1987), 1-64.
11. Cohen, M.A., Ritter, F.E. and Haynes, S.R. Herbal: A high-level language and development environment for developing cognitive models in Soar *Proceedings of the 14th Conference on Behavior Representation in Modeling and Simulation*, Orlando, FL, U. of Central Florida, 2005.
12. Haynes, S.R., Kannampallil, T.G., Larson, L.L. and Garg, N. Optimizing anti-terrorism resource allocation. *J. Am. Soc. Inf. Sci. Technol.*, *56*, 3, (2005), 299-309.
13. Fischer, G., Nakakoji, K., Ostwald, J., Stahl, G. and Sumner, T., Embedding Computer-Based Critics in the Contexts of Design. in *Human Factors in Computing Systems INTERCHI'93*, (Amsterdam, Netherlands, 1993), 157-164.