A Function-to-Task Process Model for Adaptive Automation System Design

Jason M. Bindewald[a,*], Michael E. Miller[b], Gilbert L. Peterson[a]

[a] *Department of Electrical and Computer Engineering, Air Force Institute of Technology, 1950 Hobson Way, Wright-Patterson AFB, OH 45433-7765*
[b] *Department of Systems and Engineering Management, Air Force Institute of Technology, 1950 Hobson Way, Wright-Patterson AFB, OH 45433-7765*

Abstract

Adaptive automation systems allow the user to complete a task seamlessly with a computer performing tasks at which the human operator struggles. Unlike traditional systems that allocate functions to either the human or the machine, adaptive automation varies the allocation of functions during system operation. Creating these systems requires designers to consider issues not present during static system development. To assist in adaptive automation system design, this paper presents the concept of inherent tasks and takes advantage of this concept to create the *function-to-task design process model*. This process model helps the designer determine how to allocate functions to the human, machine, or dynamically between the two. An illustration of the process demonstrates the potential complexity within adaptive automation systems and how the process model aids in understanding this complexity during early stage design.

Keywords: Adaptive Automation, Function Allocation, Man-Machine Allocation, Inherent Tasks, Interaction Design

1. Introduction and Definitions

Consumer, commercial, and government systems increasingly apply automation, particularly in systems which involve time critical decisions and actions. These systems include manufacturing plant process control (Itoh et al., 1999; Valente et al., 2010; Valente and Carpanzano, 2011), aircrew and air traffic control (Prevot et al., 2008), and remotely piloted or controlled vehicles (Parasuraman and Wickens, 2008; Parasuraman et al., 2009; Kidwell et al., 2012). Automation can improve the performance of systems without increasing manpower requirements by allocating routine tasks to automated aids, improving

*Corresponding author. Tel.: +1 937 255 3636 x6129; fax: +1 937 656 7061
Email addresses: Jason.Bindewald@afit.edu (Jason M. Bindewald), Michael.Miller@afit.edu (Michael E. Miller), Gilbert.Peterson@afit.edu (Gilbert L. Peterson)

safety through the use of automated monitoring aids, and reducing the overall cost or improving productivity of systems (Rouse, 1981). Additionally, automation can permit removal of the operator from particularly undesirable or dangerous environments (Nakazawa, 1993), increasing the safety and reducing stressors placed upon the operator.

Unfortunately, automation system designers have limited ability to project future events, and are often unable to adapt when unforeseen circumstances occur. As such, utilization of a human operator who can adapt to these unforeseen circumstances to provide system resilience is desirable (Woods and Cook, 2006). With the inclusion of a human operator, other problems arise. Some include over-reliance on automation (Itoh, 2011), placing inappropriate levels of trust in the automation (Dzindolet et al., 2003; Lee and See, 2004; Merritt et al., 2012), or losing situation awareness to preclude appropriate recovery from automation failures (Itoh, 2011). Further, as operators are not performing active control of the system, they may not practice the knowledge necessary to operate the system and can suffer from skill atrophy (Kirwan, 2005). As a result, practitioners developed adaptive automation systems to maintain user engagement, without overloading operators (Rouse, 1977).

Automation is the capability "to have a computer carry out certain functions that the human operator would normally perform" (Parasuraman et al., 2000). Knowing which entity will perform a given task helps determine whether to automate a task or not. There are many types of tasks, and consequently, several forms of automation. The categories of automation can include, "the mechanization and integration of the sensing of environmental variables; data processing and decision making; mechanical action; and 'information action' by communication of processed information to people" (Sheridan and Parasuraman, 2005).

Since, Rouse proposed a dynamic approach to automated decision-making (Rouse, 1977, 1981), the field has adopted the terms *adaptive automation* and *adaptive systems* to define the idea of an automated system that can adapt to a changing environment. Within research, the definition of adaptive automation has been subject to debate. Most authors would agree that levels or types of automation change in an adaptive system. For example, Dorneich et al. (2012) define adaptive systems as those "allowing the system to invoke varying levels of automation support in real time during task execution, often on the basis of its assessment of the current context...invoking them only as needed". This view of adaptive automation places the onus of determining the current automation state on the system. However, others have shown that even the determination of who 'adapts' the system (e.g., the system, the operator, etc.) can fall on a sliding scale (Parasuraman and Wickens, 2008).

Within the current context, a *system* is a combination of hardware, software, and human operators that work together to accomplish one or more goals. As a focus of the paper is system design, the term *machine* refers to the combination of all hardware and software within the system with which the human operator interacts.

Although the terms *function* and *task* are sometimes applied interchange-

ably (Bye et al., 1999), clear differentiation of these terms leads to a better understanding of the proposed process model. Here, we define a *function* as an action that an element of a system performs to accomplish the desired goals or to provide the desired capability. A function is delineated from a task as the function is not allocated to an entity. A *task* is a function allocated to a specific entity, and represents the actions necessary for the entity to perform the function.

A task's allocation can be either explicit or inherent. An *explicit* task is one that is directly indicated by a previously defined function. Alternatively, an *inherent* task arises only once a function is allocated to a specific entity. An inherent task is not required by the function, but is necessary to enable the allocated entity to perform the function. For example, the system might require an operator to make a selection, requiring an explicit action. However, to make this selection, the operator will need to gather appropriate information from the system or environment and make decisions, each of which are inherent tasks. *Task load* then describes the number and difficulty of tasks assigned to human operators, to which they must respond.

Workload refers to the impact of the task demand placed upon the operator's mental or physical resources. The variability in the task load imposed upon an operator (and the workload the operator experiences) originates from a number of sources. In addition to the variance of performance due to explicitly defined workload, the performance of the human operator may vary due to individual factors such as fatigue, stress level, motivation, and training level (MacDonald, 2003; Reid and Nygren, 1988).

This research presents a function-to-task design process model to aid the conceptual design of adaptive automation systems. The function-to-task design process model creates a set of visual diagrams enabling designers to better allocate tasks between human and machine. This is achieved through a set of five analysis tools allowing designers to identify points within a function network where the transitions between human and machine entities can facilitate adaptive automation. This paper proceeds as follows. Section 2 reviews the design processes currently in place for adaptive automation systems. Section 3 presents the function-to-task design process model. Section 4 illustrates the function-to-task design process model through a system design iteration. Section 5 presents conclusions summarizing the information presented.

2. Designing Adaptive Automation Systems

Discussions on the design of manned systems as a tool to aid allocation of functions or tasks between a human operator and a machine often cite Fitts' List (Fitts et al., 1951) of tasks that machines tend to perform "better" than humans and those that humans perform "better" than machines. Fitts et al. discussed tasking the machine to perform routine tasks that require high speed and force, computational power, short-term storage, or simultaneous activities; and further propose leveraging the human's flexibility, judgment, selective recall,

and inductive reasoning to improve system robustness to unforeseen circumstances. They also acknowledge the limitation of humans to correctly employ these capabilities when overloaded due to excessive task demands or to maintain alertness and employ these capabilities when not actively participating in system control.

One may consider the allocation of functions between man and machine within a system as a multi-objective optimization, wherein designers optimize some combination of performance, safety, and robustness as a function of the tasks allocated to each component. The limitations of system and human capability shape this optimization, with a significant component of human capability quantified in terms of human workload. Adaptive automation system design assumes that the number and difficulty of tasks performed will vary over time, and the tasks allocated to the human or machine need to vary to provide the human operator with an appropriate workload.

Figure 1 illustrates this concept, which depicts a two-dimensional space which arranges tasks, T1-T9, based on how well a human operator or the machine can perform them under reasonable task load. As shown, performance by either system can range from unsatisfactory through excellent (Price, 1985). We should allocate tasks, such as T1 or T8–which one entity (human or machine) can perform more satisfactorily–to the better performing entity. However, any task that either entity can perform beyond the point of satisfactory performance, we can reasonably allocate to either human or machine.

If there was no constraint on resources, one could maximize performance of the overall system by allocating tasks below the 45 degree line to the human and tasks above to the machine. However, resource constraints force a shift in the location of this line. For instance, assuming workload limits on human performance and unbounded machine resources might induce the designer to shift the dividing line lower in the plot, decreasing human workload and allocating additional tasks to the machine. On the other hand, if users' performances improve by increasing their engagement with the system, raising the dividing line allocates more tasks to the human. Therefore, adaptive automation effectively requires the system to permit this allocation line to shift up and down within this plot, allocating fewer or greater numbers of tasks to the human operator.

2.1. Automation Taxonomies

Taxonomies for adaptive automation have been proposed to accommodate the complex design space present in adaptive automation systems. Feigh et al. (2012) indicate that modifying the allocation of tasks among humans or machines can affect operator workload. However, modification of task scheduling, interaction required between the operator and other system elements, or the content of any interaction can also affect operator workload. Although not explicitly captured, these modifications may involve systems with multiple machines or multiple humans (Calleja and Troost, 2005). Considering the interaction between an individual operator and a machine, Parasuraman et al. (2000) proposed a model for describing levels of automation that builds upon the work of Sheridan and Verplank (1978) to discern between the types and levels of automation. The

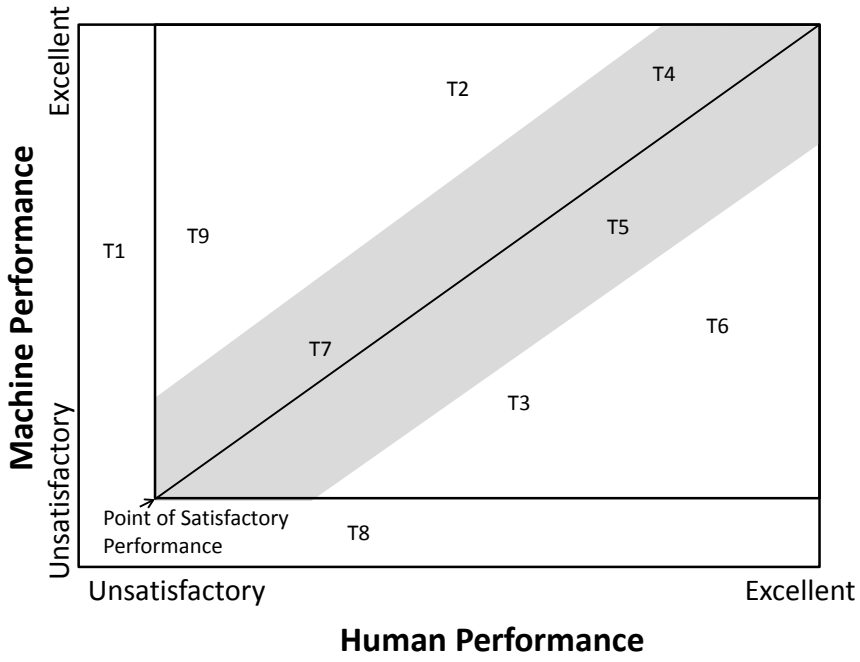

Figure 1: Diagram for task allocation in adaptive automation, adapted from (Price, 1985).

model delineates the types of tasks performed based on the four-stages of human information processing: sensory processing, perception/working memory, decision making, and response selection. Within these four stages, they expand upon the ideas of Sheridan and Verplank and codify them further into a 10-point scale describing the levels of automation, ranging from "1. The computer offers no assistance; human must take all decisions and actions" all the way to "10. The computer decides everything, acts autonomously, ignoring the human."

Endsley (1999) proposed four core human functions that a system could automate independently of one another, including: monitoring, generating alternatives, selecting alternatives, and implementing the selected alternative. This framework assigns each of these four tasks to either the human or machine (or both in some cases) and enumerates the level of automation between fully autonomous and fully human-implemented, providing a two-dimensional space over which to define automation. Each of these classification schemes permits the differentiation between intermediate levels of automation, explicitly defining which human task a given level automates. Each model aids the creation and classification of automation states for tasks the human or machine can perform, helping the system designer determine "what" to automate and "to what extent"(i.e. level of automation). Although designers can apply "level of automation" models to any system employing automation, they are important in

systems employing adaptive automation as they permit the designer to determine what part of and how to automate a task so that changes in automation level can be clearly described.

Although the adaptive automation taxonomy Feigh et al. (2012) proposes does not fully overlap the automation taxonomies provided by either Parasuraman et al. (2000) or Endsley (1999), the taxonomies are not independent of one another. Feigh et al. uniquely highlight the fact that not all tasks are time critical, and systems can reprioritize them during periods of peak workload. They also discuss the allocation of tasks between humans and machines–alluding to various levels for automation of tasks that include selection or implementation of alternatives. Additionally, they contend that automation of generating alternatives and monitoring requires automatically generated information displayed to the human operator, forcing a change in the interaction and content of interaction. Each of these methods, therefore provides a different way to classify and consider the effect of changes in autonomy on operator workload.

2.2. Human-Machine Interaction

The need to provide effective communication between the human and machine impedes human interaction with automated systems. In some cases–such as flight control automation–the design of this interaction can have life-or-death consequences (Geiselman et al., 2013; Kaber et al., 2001). Unfortunately, this interaction can become increasingly complex in systems employing adaptive automation. William Rouse's analysis of human-machine interaction within a dynamic system is a seminal article in this field (Rouse, 1981). Rouse shows the different forms of communication with the system as a set of five interaction loops. The first two loops, in which it is possible that no communication is required, represented manual control and completely automated control. In the third loop, wherein he coins the term *overt communication*, the human and machine operators of a system directly communicate information about their tasks. The human operator must take explicit actions to control the machine, and the machine must explicitly provide information. The human operator must consciously read, listen to, or otherwise receive this information. The last two loops represent more subtle communication which typically occurs among humans; *covert communication*, with the fourth loop representing covert human to machine communication and the fifth covert machine-to-human communication. Information communicated indirectly–which might include state information–characterizes covert communication. The timeliness of a response from a teammate, where hesitancy in response signals uncertainty and fast authoritative response indicates certainty, provides an example of covert communication.

Unfortunately, communication errors occur between human operators and machines as the machine can fail to communicate critical state information, the information leading to the selection of a critical state, or less direct information, such as the certainty of this information (Geiselman et al., 2013). Recent research focuses on improving covert communication from the human to the machine through the use of psychophysiological measures, such as electroencephalography

(EEG), electrocardiography (ECG) , electrodermal activity (EDA), electromyography (EMG) (Dorneich et al., 2012; Byrne and Parasuraman, 1996; Haarmann et al., 2009) or behavioral measures, such as eye gaze patterns. Such measures have the "potential to yield real-time estimates of mental state" (Byrne and Parasuraman, 1996), thus allowing the machine to gain information regarding the state of the human operator.

The infeasibility of communicating all automated tasks from a machine to a human aside, the human in an automated system requires enough information to permit appropriate situation awareness. Since the human operator assumes control in the event of a mishap or in order to make a critical decision, the human needs an understanding of the current system and environment state. Several research efforts devote effort toward finding an appropriate balance between providing enough information for situation awareness and overloading the human operator with information (Wickens, 2008; Endsley, 1999; Kaber et al., 2001; Sheridan and Parasuraman, 2005; Manzey et al., 2012; Parasuraman et al., 2008). Further, all communication will affect the user's workload, potentially resulting in overload conditions. However, the relationship between how humans attend to, receive, process, and act upon information creates complexity, and the interaction influences the human operator's perceived workload (Wickens, 2008).

The manner in which feedback is given influences the resulting system. For example, Manzey et al. demonstrate that users are much more likely to develop a proper level of trust with a system when the system gives them negative feedback loops rather than positive ones (Manzey et al., 2012). Further issues, such as how to design a system to manage interruptions in a socially acceptable manner and analyzing the positive and negative consequences of automating the interruption management task (Dorneich et al., 2012), are important. The idea of etiquette flows naturally into the concept of trust, directly impacting the human operator's trust of the system.

While the design of the human-machine interface can be complex, this interface requires an understanding of the information that the human and the machine must communicate to facilitate task completion. The importance of this information necessitates its presentation in a way that does not overload the operator and recognizes the fact that the human operator will not necessarily receive all information the system provides.

2.3. User-centered Design and Task Analysis

User-centered design (Norman and Draper, 1986) has evolved to aid the design of systems including a significant user interaction component. This process often involves the steps of: 1) establishing a vision for the system, including an initial system concept and business objectives; 2) analyzing requirements and user needs to understand how users perform the tasks within the boundary of the system concept and the context of use; 3) Designing for usability through conceptual and detailed interaction design, including prototyping; 4)evaluating the system, which can involve early focused deployment and evaluation of the system; and 5) applying learnings from the evaluation to provide feedback and improvements to the overall system design (Gulliksen et al., 2003). Each of the steps in this

process, as well as the overall process, are conducted iteratively until a desired level of usability is attained. During the development of complex systems, this process can involve individuals from numerous disciplines, to include systems engineering, human factors, software design, human computer interface design and information system management, all of whom can apply different design and evaluation techniques during the vision development, analysis, and design phases.

Tools for capturing the requirements for a design within the systems engineering community include forms of the structured analysis and design technique (SADT) (Ross, 1977), as well as the Systems Modeling Language (SysML) (Delligatti, 2013). SADT primarily focuses on the documentation and decomposition of the process to be employed by the machine during design. SysML was developed from the Unified Modeling Language (UML) (Larman, 2004), which was originally developed for the design of software systems. SysML includes a number of tools to capture and communicate the vision for the system, analyze requirements, develop conceptual and eventually detailed designs, and to associate test procedures and outcomes, permitting verification and validation of the system requirements. These tools can be applied in either a descriptive fashion, describing an existing system when analyzing requirements, or in a prescriptive fashion, documenting vision, requirements, and design of the system under design. While these tools focus primarily upon design of the machine, certain tools, including use case and activity diagrams can capture human interaction with the system.

Within the human factors community, Hierarchical Task Analysis (HTA) (Annett and Duncan, 1967; Annett, 2003) is commonly applied to systematically capture and decompose human activities to describe processes that are commonly applied by the human. Information from these analyses can be depicted in a number of forms, one method of particular interest is the Operational Sequence Diagram (OSD) (Blanchard and Fabrycky, 2006). The OSD captures the flow of information between a human and machine, indicating the timing, direction and modality of information flow during each exchange between a human and the machine. More recently other task modeling languages have been developed. These tools include the ability to model tasks performed by groups of individuals, for example Groupware Task Analysis (GTA) (Van Welie and Van Der Veer, 2003), as well as methods to assess differences between human and machine knowledge structures, e.g., Task Knowledge Structures (TKS) (Johnson et al., 1988). UML or SysML tools, can be applied to model the results of a task analysis. For example, use cases may represent an informal task analysis structure (Paterno, 2001) and activity diagrams can capture tasks as well as information flow between entities (Lu et al., 2003).

Besides these tools, task description languages extend descriptive task analysis to provide prescriptive tools useful in specifying the design of the human interface. The Goals, Operators, Methods, and Selection Rules (GOMS) method (Card et al., 1986) attempts to define the system through a set of goals, decomposing goals, determining how users perform tasks to achieve these goals and assessing the method of interaction on user performance. Other tools, such as

DIANE+ (Tarby and Barthet, 1996) are useful during specification and user interface design (Vanderdonckt et al., 1998). ConcurTaskTrees (CTT) (Paternò et al., 1997) uses a hierarchical task structure with a focus on defining a number of different types of temporal relationships between tasks to aid the design of the user interface.

These tools can be geared towards certain application environments. For example, AMBOSS (Giese et al., 2008) is primarily relevant to understanding human error in safety critical systems. Similarly the Functional Resonance Analysis Method (FRAM) (Hollnagel, 2012) is primarily concerned with utilizing performance variability as it pertains to achieving desirable (or dampening undesirable) outcomes. Tools have also been developed in the management information field for capturing and designing workflows in complex business processes. Examples of these workflow languages are Yet Another Workflow Language (YAWL) (van der Aalst and ter Hofstede, 2005) and Web Services Business Process Execution Language (WS-BPEL) (Jordan et al., 2007).

In the context of the current research, a task model of particular interest is HAMSTERS and recent extensions to this model (Martinie et al., 2011, 2013). Similar to CTT, HAMSTERS provides notation for indicating machine tasks separately from human tasks when decomposing a goal or high level function (i.e., abstract task in the HAMSTERS nomenclature). Each of these modeling languages additionally provide notation for interactive tasks, tasks performed by the human or system to facilitate interaction between the two entities. By creating different hierarchical networks with different tasks allocated to human or machine, these methods can be used to assess various static automation strategies. Further, HAMSTERS has recently been extended to permit the modeling of not only the hierarchical task relationship but the temporal sequencing of these tasks.

Existing task modeling tools are primarily designed for systems with static rather than adaptive automation. As a result, many of these tools do not provide tools to move across the function-task distinction described previously. In their designed context, these tools and methods typically do not explicitly provide ways to move from functions with an unassigned operational entity (e.g., machine or human) to tasks with an operational entity assigned. Therefore, a process model with a task model is proposed that aids the designer when determining functions to statically allocate to humans and machines, as well as functions to dynamically allocate between the human and machine.

3. Function-to-Task Design Process Model

This section presents the proposed process model for allocating functions to entities (e.g., human or machine) that leads to adaptive automation allocations. Figure 2 graphically depicts the function-to-task design process model. The function-to-task model usually proceeds in a linear fashion, as indicated by the solid bold arrows in Figure 2. In some cases, completing steps in the process will force the design back to a previous step for revision; likely locations for revision steps are indicated by the dashed arrows in Figure 2. It is likely that a proper

decomposition of the system will not occur on the first attempt and, therefore, this process can become iterative.

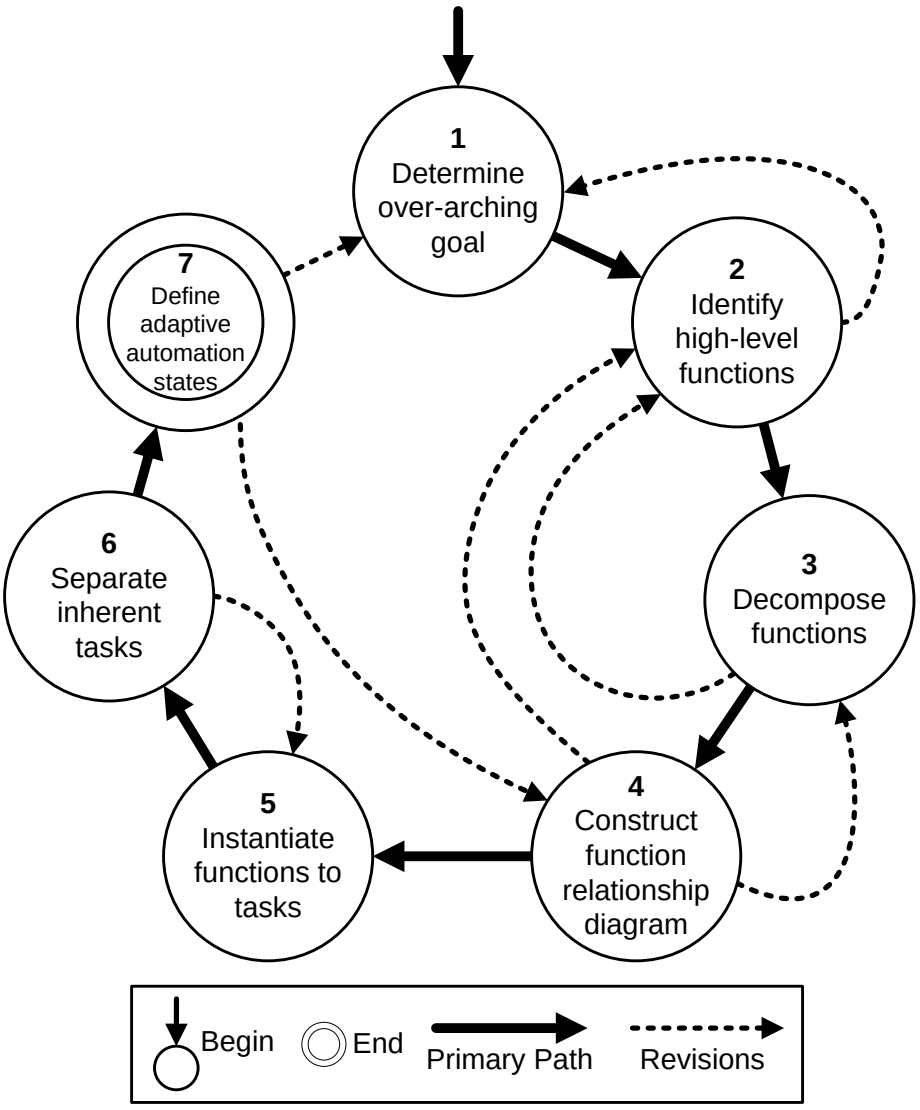

Figure 2: The Function-to-Task Design Process Model, depicting the developmental flow and typical revision loops necessary for design refinement.

3.1. Step 1: Determine Over-Arching Goal

In the first step, the designer determines the goal(s) of the system. The over-arching goal should answer the question, "What is the system trying to

achieve?" Any predetermination as to how the task must be accomplished should be excluded. For example, a goal to "obtain milk through a purchase," contains no pre-conceived notion of *how* to purchase the milk. The overall goal should be distilled to only its essential elements–those requirements that are unavoidable without making the scope over-expansive. For example, obtaining milk is a less exclusive goal than purchasing milk. However, broadening the goal beyond solutions under serious consideration is counter-productive (e.g., we would not expand the goal of purchase milk unless we would consider alternate methods of obtaining milk).

3.2. Step 2: Identify High-Level Functions

The second step is to identify the functions that must be performed to achieve the goal(s). The question to answer at this stage is, "How do we achieve the over-arching goal?" The functions at this stage should be high-level, and, depending on the goal, could consist of only one or a small number of functions. At this point, the designer has not allocated these functions to performing entities. Therefore, the high-level functions must be defined such that they can be allocated to any available entity.

3.3. Step 3: Decompose Functions

Functions are composed of sub-functions in a modular or hierarchical fashion. The high-level functions from step two are decomposed into sub-functions until they reach the atomic function level. *Atomic functions* are functions that can only be performed by a single entity (e.g., it cannot reasonably be decomposed into more than one function where one or more of these functions would realistically be allocated to a human and another would be allocated to a machine). Further decomposition of an atomic function is not desirable, making the determination of automation state a discrete decision. Consequently, all functionality for which the system must account falls under a high-level function. The complexity of a given function depends on the number and interrelationships of its sub-functions.

All non-atomic functions are composed of lower-level functions. There are many proposed methods for decomposing a function, including methods from structured analysis, such as Integrated Computer Aided Manufacturing Definition for Function (IDEF) Modeling (Buede, 2011). The designer should perform decomposition until functions are indivisible between multiple entities, resulting in *atomic functions*. In practice, decomposing each function to the point where it is indivisible is not necessary, but instead the designer should decompose each function to the point at which it is impractical to allocate a portion of a function to two separate entities. With system evolution, it may be necessary to readdress the function decomposition as functions which are impractical to allocate to separate entities may change as technology evolves.

The actions taken in step three repeatedly address the question, "Can more than one entity perform function x?" For the purposes of system representation, step three should produce a set of nodes. Although graphical depictions of the atomic functions (such as IDEF diagrams that maintain knowledge of the

hierarchical decomposition (Cheng-Leong et al., 1999)) are useful, one must take care when naming the functions to insure that all unique functions have unique names.

Steps one through three are similar to the hierarchical task breakdown of methods such as HTA or GOMS but do not include allocation of the detail of user interaction inherent in an allocated function. For example, atomic functions are similar to the leaf nodes created in a hierarchical task decomposition, but functions remain unallocated between human and machine. Therefore, the designer should take special care to ensure an allocation-free breakdown during the first three steps of the present model.

3.4. Step 4: Construct Function Relationship Diagram

After identifying the atomic functions, the functions are transformed into a network by exploring the relationships between the atomic functions. To complete a function, a subset of its atomic functions must be completed in a pre-determined order, as information generated by a function will be an input to other functions. Further, it is common for an atomic function to reside within multiple function hierarchies. These relationships are depicted in a *function relationship diagram* (FRD) wherein the atomic functions represent nodes and the information to pass between nodes are represented by the connecting arrows. The resulting network provides a temporal ordering similar to that introduced for HAMSTERS (Martinie et al., 2013) for a task network. A function is either mandatory (represented by a solid border in the FRD) or optional (represented by a dashed border in the FRD). Optional functions are those that may need to be performed within some task instances, but not others. Figure 3 shows a legend of the different structures used in the FRD.

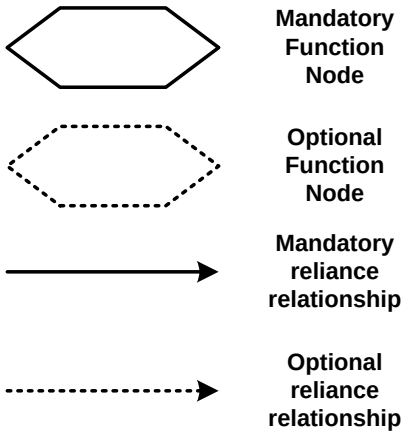

Figure 3: Legend of notation used in Function Relationship Diagram (FRD) structures.

A large number of temporal relationship types can exist between functions. However, each pair of atomic functions can have two possible relationships:

dependent or independent. Dependent relationships arise when the completion or product of one task directly influences the other. Dependent relationships are represented by an arrow in the FRD. A function is independent of another when it has no reliance or influence on the completion of the other. All non-connected functions are independent from each other.

Similar to the functions themselves, a relationship is either mandatory (represented by a solid arrow in the FRD) or optional (represented by a dashed arrow in the FRD). A mandatory relationship implies that one function necessarily leads to the next, while an optional relationship implies a set of circumstances that could avoid this relationship.

Using mandatory and optional functions and relationships, several more-complex relationships, such as those referenced in (Paternò et al., 1997), can be represented. For example, interleaved relationships can be represented as two separate flows within an FRD, with the two functions each with an optional relationship with another common node. An iteration relationship is represented by an optional relationship arrow looping the function node back on itself. The complex relationships are not represented directly, because decomposing these relationships to a lower level allows for further freedom in design.

Multiple relationships may flow into or out of a given node. If an atomic function does not connect to other atomic functions from the higher-level function from which it was derived, the function decomposition should be re-addressed, as this condition violates the rules of the function decomposition. The diagram at this point should not involve the instantiation of function performers (i.e. it is still a *function* relationship diagram and not a *task* relationship diagram).

3.5. Step 5: Instantiate functions to tasks

In step five, the system designer allocates each function to an entity: human or machine. Specific instances of humans and machines are not assigned, we are concerned only *that* a human or machine is performing the function, *not which* human or machine performs it. In fact, to simplify the current discussion, it is assumed that only one human operator is present in the system under design, although this concept could be refined to permit the inclusion of multiple humans interacting with one or more interconnected machines. This step sets a baseline for the states of automation. A *task relation diagram* (TRD) that demonstrates the flow of information from one entity to another results from this step.

The first step in task instantiation involves induced assignments. Some constraint may mandate the instantiation of a specific function, or set of functions, to a specific entity. Induced assignments can come from rules, capabilities, available resources, or other avenues, but must be addressed no matter the reason for their inclusion. These are assigned first, before any other instantiations are made. Examples of induced functions include decision nodes in systems where humans hold final decision authority or a complex calculation that a human is incapable of performing and a machine must perform.

Once the induced assignments are made, the designer can address the more flexible assignments. The adaptive automation task allocation model discussed

in 1 enables the determination of which tasks to assign to a human or machine. By using the model demonstrated in Figure 1, tasks can be assigned to the entity capable of performing the function with maximum proficiency. Although this model may provide insight into which function nodes to instantiate to which entity, it can also draw attention to nodes that are not clearly favored to one entity or the other. The resulting TRD should indicate each node as human or machine, as shown in the legend contained in Figure 4.

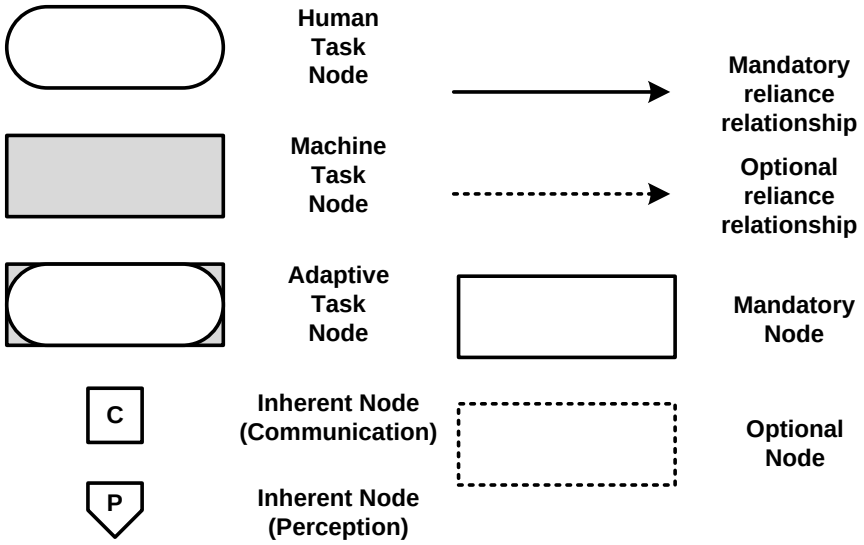

Figure 4: Legend of notation used in Task Relationship Diagram (TRD) structures.

3.6. Step 6: Separate Inherent Tasks

At this step, all functions have been allocated to entities. The resulting TRD consists only of explicit tasks the atomic functions specify. Completion of the task allocation consists of specifying the inherent tasks. Inherent tasks are those tasks that present themselves as the product of a specific task instantiation. However, these inherent tasks can also result from the interactions between the explicit tasks or specific resources available to the system.

The information exchange between entities during a task handoff is the primary source of inherent tasks. Once the designer assigns a task to a machine or human, a set of new complexities emerge through the task relationship diagram: task handoffs. There are four types of task handoffs possible: human to human, human to machine, machine to human, and machine to machine. For the purposes of an adaptive automation system, task handoffs that cross between human and machine are important. A human-to-machine or machine-to-human task handoff requires two inherent tasks that are not present in the underlying

functions: *communication* of the information by the losing entity and *perception* of the information by the gaining entity.

Communication of information requires the current performing entity to format the information such that the next entity understands it. For example, a machine that just completed a movie recommendation search task must ensure that it communicates the recommended films to the human before the human can complete the subsequent movie selection task (i.e., displays this information on a screen). On the other end, perception involves the next task performer's ability to obtain and interpret the information communicated to permit subsequent task completion. It's important to make the inherent task nodes in the task relationship diagram visually distinctive. This distinction permits complex task relationships to become more apparent. Step six produces a complete TRD similar to that produced by the previous step, but including both explicit and inherent tasks. As shown in the legend (Figure 4), the communication and perception tasks of an inherent task are represented by the "C" and "P" nodes that will appear in pairs.

At this point, the designer may find it useful to reiterate through the process to ensure that the diagram truly represents the desired process and system. After this stage, an initial allocation exists. Modeling or prototyping tools can then be used to determine if the human or humans assigned to operate within the system are capable of performing the tasks required from them during typical system operation while having high enough workload to remain engaged with the system. If not, steps five and six are revised until the design attains a desired level of workload. To reduce workload, for example, the human can give a task involved within a complex relationship within the TRD to the machine.

3.7. Step 7: Define Adaptive Automation States

The inherent tasks of communication and perception provide one of the most important steps in designing an automation system. When the designer adds adaptive automation to the system, an understanding of cognitive task handoffs is crucial. The selection of a set of atomic tasks, or groups of tasks, in the TRD to become adaptive nodes makes the automation adaptive, and consists of identifying nodes that can switch between human and machine instantiation based on some pre-defined trigger.

Selecting a given node as an adaptive node, adds complexity to the overall system. As a node switches from static human or machine to adaptive, the number of handoff types needed can double for each outgoing connection. For example, a human node connected to a machine node requires one type of handoff, a human node connected to an adaptive node requires two, and an adaptive node connected to an adaptive node requires four. It should be noted, however, that this added complexity only occurs for nodes that are independently switched within the design. It is possible for a group of nodes to always be switched in concert with one another as a cluster, in which case, only the number of connections between this group of nodes and the nodes they connect with are doubled.

Step seven finalizes the selection of adaptive nodes. As shown in the legend (Figure 4), adaptive nodes are represented by a shape that combines those of the human and machine nodes. Choosing adaptive nodes is ultimately a subjective task. However, an analysis of the TRD can help make these decisions easier and more grounded. Five analysis tools include: determining the number of possible states, node clustering, task handoff analysis, branch counting, and inherent task load comparison. By iterating through these tools, an adaptive automation system emerges.

3.7.1. Number of possible states

Once the designer selects adaptive nodes, they must readdress the complexity and handoffs created through the selection. One way to assess complexity involves determining the number of possible automation states. For each independently adaptive automation node in the relationship graph, two possible states exist: human and machine. Therefore, the number of possible states equals 2^x, where x is the number of adaptive nodes in the current design. Fewer independently adaptive nodes means exponentially fewer possible states. Within the TRD, one can simply count the number of desired adaptive nodes and plug it into the above equation. This is a rough approximation of the potential challenges.

3.7.2. Task handoffs

Related to the total number of possible states, one way the TRD can help analyze the designed system is through an analysis of the task handoffs–specifically the number of different-entity handoffs. In each possible case where a machine hands off to a human or human to a machine, count one handoff. In the case where a node is set as adaptive, this implies that a handoff from an adaptive node to another adaptive node counts twice, while an adaptive to non-adaptive node handoff will count once. This handoff count suggests the number of nodes where task load shifts from one entity to another. By focusing on these nodes, potential bottlenecks appear due to certain handoff tasks taking place more often than at other locations. Highlighting these tradeoffs allows the system designer to visualize the locations where inherent tasks–specifically those associated with communication and perception, as described in Section 3.6–reside.

3.7.3. Node clustering

Functions clustered based upon the degree of the edges in and out of a given node tend to provide similar or highly inter-related functions. Therefore, automation of a cluster as a set can often be achieved with greater effect than just automating one function in the set. Conversely, the designer could also instantiate all of the tasks in a cluster to a human, since the information associated with the multiple edges will not need to be exchanged.

An example of this would be in piloting an aircraft. Although the "takeoff function" contains many lower level functions, there are many complex groups of functions within it that naturally group together to ensure a proper amount of situation awareness, where the necessary information is provided through the right kinds of feedback.

Within the TRD, an adaptive node cluster can be represented by a large adaptive node surrounding a set of tasks. This implies that all of the tasks within the adaptive node cluster will all always have the same allocation. This allows for adaptation without increasing the number of states exponentially. Additionally, since a hierarchical notation was not kept in Steps 1-4, the TRD now has flexibility to cluster in unique ways not easily perceivable through the hierarchical breakdown.

3.7.4. Branch counting

Branch counting refers to the idea of determining the number of other atomic tasks to which one specific atomic task connects. A task that influences or is influenced by a large number of tasks makes automation more difficult. This is because changing a node to adaptive requires an inherent communication/perception node to each incoming and outgoing relationship. For example, forming a node with one outgoing relationship and one incoming relationship adaptive creates two new communication/perception nodes, while doing the same to a node with one incoming relationship and four outgoing relationships creates five new communication perception nodes.

Tasks that have large branch counts can often make good candidates to be members of an adaptive node cluster. Conversely, single branches within a TRD can often indicate good locations to place adaptive nodes, as the inherent task load is likely smaller. Although some individual task handoffs may be very difficult, branch counting a TRD provides a good snapshot of where there will be a large number of task handoffs that the designer may not have foreseen.

3.7.5. Inherent task load comparison

An inherent task load comparison provides another means to analyze the effectiveness of design options. This consists of a comparison of the relationship diagrams created when the TRD instantiates one function as a human task versus when the TRD instantiates the same function as a machine task. The primary difference as a result of this reallocation is the number of inherent tasks that are added or subtracted. A comparison of the two instantiations helps to visually communicate inherently understandable design decisions.

4. Function to Task Process Illustrated

Space Navigator is a tablet computer based route creation game for use in adaptive automation research, shown in Figure 5. The game operates as follows. Four stationary planets are present on the screen. Each planet is one of four colors: red, green, blue, or yellow. Spaceships appear at a set interval from a random side of the screen. Each spaceship is red, green, blue, or yellow. The player must direct each spaceship to the destination planet of the same color by drawing a trajectory line on the game screen using their finger. The spaceship then follows this line at a constant rate. Spaceships continue to appear until an allotted time of five minutes is over. If desired, trajectories may be re-drawn, to avoid a collision and account for dynamic changes.

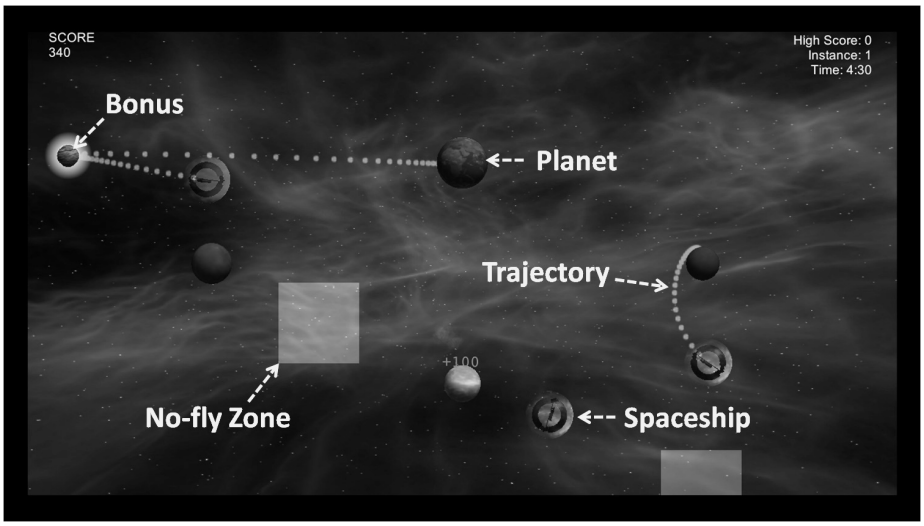

Figure 5: Screen capture from a game of Space Navigator, pointing out spaceships, planets, trajectories, bonuses and no-fly zones.

Points are earned when a ship successfully reaches its destination planet or traverses any of a number of small bonuses that appear throughout the play area. Upon reaching its destination planet, a spaceship disappears from the screen. When spaceships collide, points are lost and each spaceship involved in the collision is lost. Additionally, points are lost when a spaceship traverses one of several "no-fly zones" (NFZs) that move throughout the play area at set time intervals.

The object of the game is to earn as many points as possible in five minutes. The player's task load varies with changes to both the rate at which spaceships appear and the number of no-fly zones. Figure 5 shows a screen capture from a game of Space Navigator, which illustrates various elements of the game.

The function-to-task process model is illustrated here by designing an adaptive automation system to aid a user during play of Space Navigator. The result of the process is an allocation of tasks to aid adaptive automation system design. The dynamism of elements within Space Navigator (e.g. movement of no-fly zones, random appearance of new ships, etc.) do not allow for creating an "optimal" automated player. Because the game is relatively simple in its mechanism while still difficult for a machine to perform optimally, Space Navigator is a good environment for illustrating the process model. Each of the seven steps of the Function-to-Task Design Process Model (Figure 2) is addressed in the following sections.

4.1. Step 1: Determine Over-Arching Goal

To fulfill Step 1, we ask the question "What are we trying to achieve?" The goal of a *Space Navigator* player is to *score the most possible points*. In the

present case this is a simple task, since the goal is clear from the rules of the game. However, in other situations a goal may be more difficult to define. For this reason, the loop from Step 2 to Step 1 may provide further insight and refinement for more complex systems.

4.2. Step 2: Identify High-Level Functions

With the goal in hand, we ask "How do we score the most possible points?" This helps identify the high-level functions as the manners in which points change. Four high-level functions become apparent:

1. Move spaceship to intended target planet.
2. Pick up bonuses.
3. Avoid collisions with other spaceships.
4. Avoid traversing no-fly zones.

4.3. Step 3: Decompose Functions

Answering the question, "Can we further divide function x?" allows further function decomposition. After applying this decomposition we obtain the following list of atomic functions:

1. Function 1: Move spaceship to intended target planet.
    - Determine the best ship to draw route.
    - Identify destination planet of selected ship.
    - Identify if ships have routes already.
    - Create a set of possible routes.
    - Select a route.
    - Draw a line from selected ship to destination.
2. Function 2: Pick up bonuses.
    - Identify all available, non-selected bonuses.
    - Identify destination planet of selected ship.
    - Determine if route change to pick up bonus is worth points gained.
    - Determine if selected ship has a route already.
    - Adjust route to pick up bonus.
3. Function 3: Avoid collisions with other spaceships.
    - Detect likely collisions.
    - Identify destination planet of selected ship.
    - Determine if selected ship has a route already.
    - Determine if route change to avoid collision is worth points gained
    - Adjust route to avoid collisions.
4. Function 4: Avoid traversing no-fly zones.

- Identify no-fly zones.
- Identify ships headed toward a no-fly zone.
- Identify destination planet of selected ship.
- Determine if selected ship has a route already.
- Determine if no-fly zone traversal is worth lost points.
- Adjust route around no-fly zone.

This atomic function list demonstrates two concepts previously discussed in Section 3.3: the overlap of specific atomic functions and the circumstance-specific nature of atomic functions. Some atomic functions in the above list appear in multiple locations within the hierarchy. For example, the atomic function "identify destination planet of selected ship" appears in all high-level functions. It is the same function and named the same in every case. Secondly, a practitioner may consider the atomic functions listed above as more complex depending on the interpretation of the process. For example, under the avoid no-fly zones high-level function, the atomic function "Determine if no-fly zone traversal is worth lost points" could be considered a non-atomic function made up of sub-functions such as "determine the number of potential points lost," "determine amount of time added," "determine increased collision likelihood," etc. However, the designer must ask whether they would consider dividing these tasks among human and machine entities or whether they would always assign them to the same entity. An important note from Step 3 is that moving further along the Process Model may provide insight into the proper level to end Step 3. This is represented explicitly by a revision loop from Step 4 back to Step 3 in Figure 2.

4.4. Step 4: Construct Function Relationship Diagram

We now produce the functional relationship diagram by analyzing each unique atomic function in relation to all of the other functions and assigning relationships (dependent or independent) based upon the transfer of information from one function to another. The end result of this relationship mapping is the function relationship diagram shown in Figure 6. The creation of this diagram illustrates overlapping atomic function instances, optional versus mandatory functions, and the flexibility provided for structuring the functions by removing the hierarchical structure.

In this step, there are a few instances where multiple higher-level functions contain the same atomic function (e.g. "identify destination planet of selected ship"). Only one node in the functional relationship diagram represents these functions. However, these functions interact with many different functions. The "identify destination planet of selected ship" function directly influences three separate functions. Therefore, functions that overlap multiple higher-level functions provide potential bottlenecks in the relationship diagram. That is, the information these functions produce must be available to any human or machine entity to permit subsequent functions' performance.

Optional and mandatory functions and relationships are all represented. For example, we need to identify potential collisions (mandatory function) and must

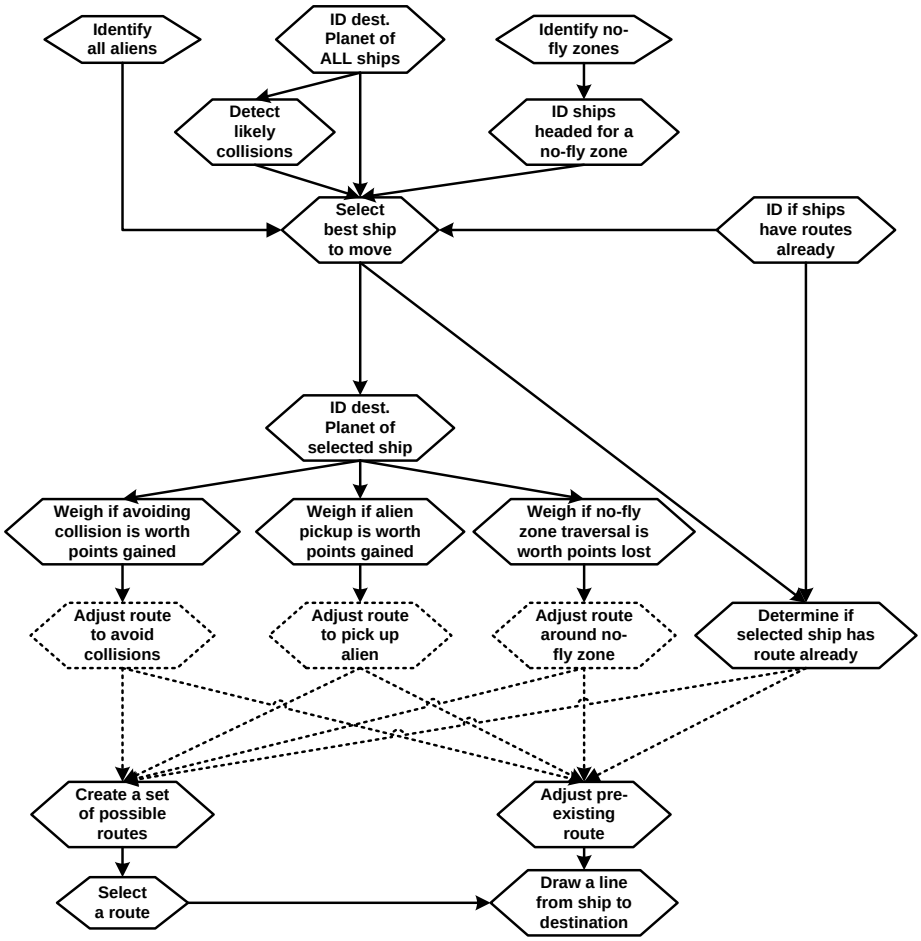

Figure 6: Functional relationship diagram of the Space Navigator game. See Figure 4 for a legend of notations used.

do it before adjusting our route for them (mandatory relationship), but this information may not necessitate a route adjustment (optional function). Then the ensuing adjustment is made in one of two ways depending on whether the selected spaceship already has a route (optional relationship).

Relationships that seem compartmentalized in the function decomposition can appear highly interconnected in the relationship diagram. The four high-level functions identified for the Space Navigator game are separated distinctly in the functional decomposition in Step 3, within the hierarchical structure. However, when the hierarchical structures are removed, the sub-functions provide a system that cannot be easily divided along the lines of the previously defined high-level functions. This change in perspective can also influence a different understanding of the high-level functions themselves. For example, looking at the resulting FRD

a designer could potentially conclude that the tasks of "spaceship selection" (the upper half of the diagram), "action decision" (the bottom half of the diagram), and "action performance" (the final function) are the higher-level functions. This demonstrates how the FRD provides a revision path that can lead the practitioner back to Steps 2 or 3 for further analysis.

4.5. Step 5: Instantiate functions to tasks

The design goal in this example is to apply automation to aid the user when interacting with this game where the assumed default state is that the human operator will perform all functions. Therefore, the goal of the function allocation in this particular example is to identify alternate automation states, which will permit an operator to perform well in the presence of exceptionally high spaceship spawn rates. Referring to Figure 6, one can see that the functions in the center of the diagram are highly inter-connected. This interconnection implies that the human and machine would need to exchange significant amounts of information if elements within this region of the figure were divided between these entities. However, other elements near the periphery of the diagram are not as highly interconnected. As a result, allocation of many of these elements to the machine are likely to result in less need for communication between the human and machine.

Based on this analysis and the performance of the human and machine, Figure 7 represents a potential task allocation and resulting task relationship diagram. In the diagram, tasks that the machine controls are those that appear as gray boxes and those that the human controls are represented by rounded boxes. Note that Figure 7 shows a TRD after Step 6 is complete.

4.6. Step 6: Separate Inherent Tasks

As discussed earlier, the "C" and "P" nodes are also shown in Figure 7. These nodes indicate the need for the entity performing the function near the "C" node to perform the inherent task of communicating to the receiving entity and the need for the entity performing the function near the "P" node to perform the inherent task of perceiving and interpreting the information to enable performance of the function. Because of the selected function allocation, there are several task handoffs from human to machine and vice versa as the "C" and "P" nodes indicate. Some of the communication/perception chains are inconsequential, like communicating from the machine to human if a specific spaceship has a route already–as a simple path is already drawn between entities to aid transfer of this information. However, others are more difficult.

For example, communicating the destination planet for all ships to a human can be simple, but it is important and perhaps difficult to ensure perception. If the ships are color-coded to align with a specific planet, this task is simple for most people when few entities are available. However, it becomes increasingly difficult as the number of entities increase and in some cases impossible for certain individuals (e.g., those who are color blind). Therefore, it is not only needed to communicate the information, but to confirm the transfer of critical

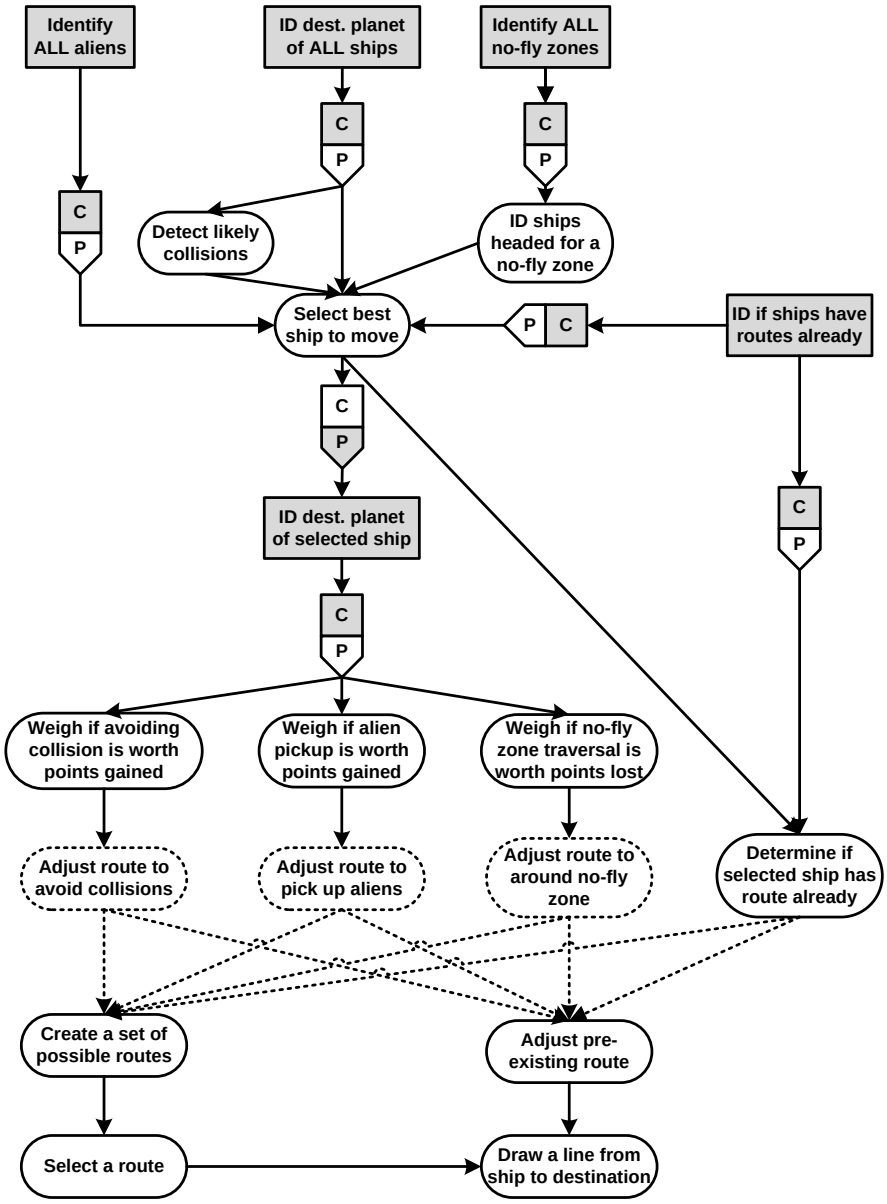

Figure 7: Task relationship diagram of the Space Navigator game with inherent tasks shown, where functions have now been allocated to human and machine. See Figure 4 for a legend of notations used.

information to insure a handoff. Steps 5 and 6 are interconnected. The Process Model (Figure 2) has a revision loop connecting the latter to the former, as the inherent tasks that appear in Step 6 can inform decisions in Step 5.

4.7. Step 7: Define Adaptive Automation States

Finally, we apply adaptive automation to the TRD. Figure 8 shows how the process changes after adaptive automation nodes are selected. Although all of the tools presented in Step 7 are useful, some will prove more useful in specific situations than others. For *Space Navigator*, the number of possible states and task handoffs are implied by application of the other tools: node clustering, branch counting, and inherent task comparison.

Looking at the original TRD in Figure 7, a few groupings of tasks begin to appear. One shows up near the top that encapsulates several spaceship selection related tasks, while a second group near the bottom represents several action-oriented tasks. These groups of tasks could prove useful as adaptive node clusters.

Branch counting is then applied to narrow the results from our node clustering. The action-oriented cluster is eliminated due to the large number of branches present in this location of the diagram. From a more subjective perspective, it would obviously be difficult to automate these tasks as they tend to rely on dynamic elements of the game (e.g. no-fly zones and ships that move).

The final TRD in Figure 8 arises from inherent task load comparison. The spaceship selection cluster is further refined by analyzing where the inherent tasks cross. By looking at these tasks, we see that there is a group of tasks that are all controlled by the same entity. Thus allowing a set of similar communication and perception inherent tasks. The large cluster then is chosen for an adaptive automation.

In this case, one can imagine a system that adapts to the player's workload to automate the selection of which ship to act on when the user is inundated with choices, by highlighting a specific ship. This arrangement permits the system to decide which spaceship is most important to route or reroute and conveys this information, ideally in a very clear fashion, to the human. The human selects and draws the path. As such, the system determines the critical ships to address, a task that can be difficult for the human when the screen is cluttered, while allowing the human to select a route, a task that is too complex for the machine to perform reliably. It is important to note the revision loops in the process model once again here. Once complete, the design may require a complete overhaul or several minor tweaks.

5. Conclusions

A function-to-task design process model has been presented that assists an adaptive automation system designer in determining the allocation of tasks between the human, machine, and dynamically between the two. This process model permits the designer to investigate the effects of allocation on explicit and inherent task load for the intended user when interacting with an adaptive automation system. The process model demonstrates that reallocation of functions imposes a change in inherent tasks to permit the proper communication and perception of information between the performing entities. As such, reallocation

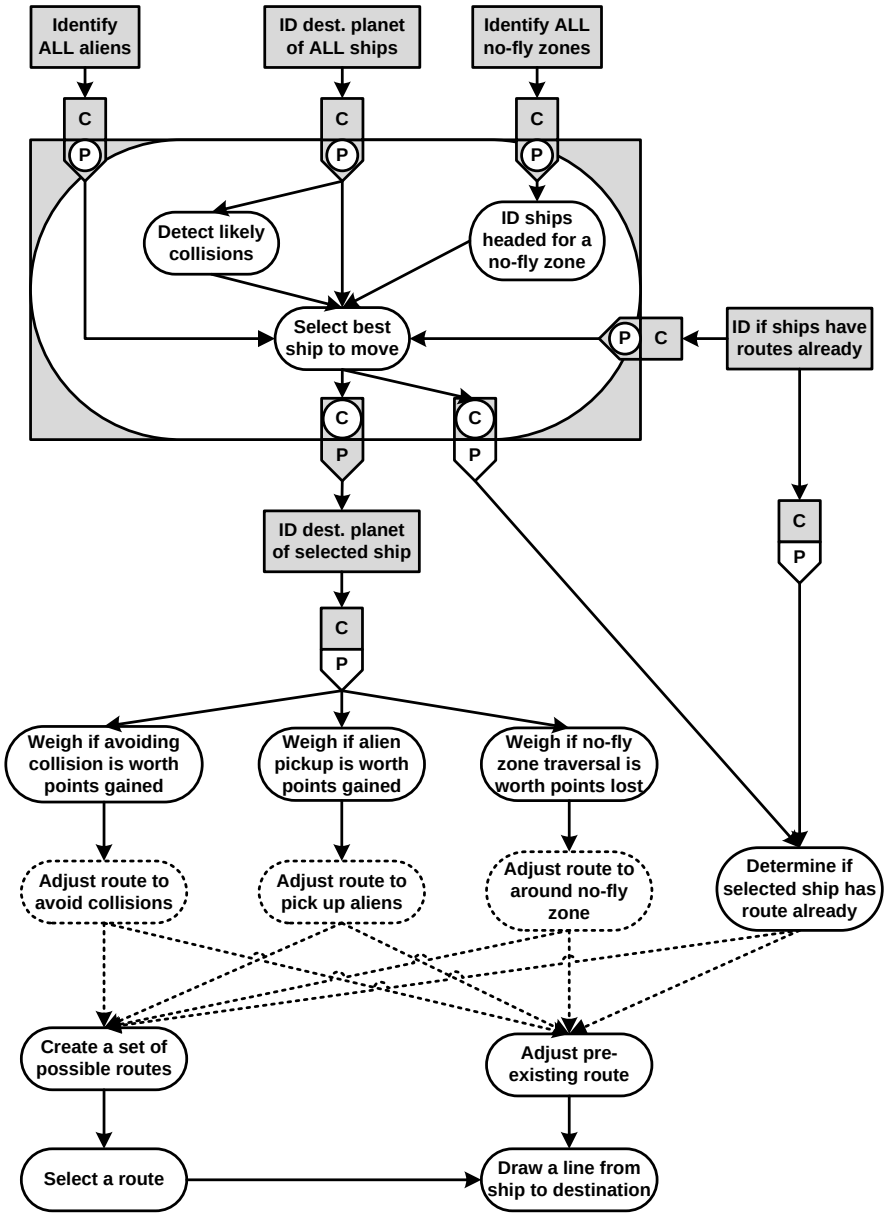

Figure 8: Task relationship diagram of the Space Navigator game, where functions have now been allocated to human nodes, machine nodes, and adaptive nodes. See Figure 4 for a legend of notations used.

of a function implies a change in information flow between the machine and human, and this change requires the allocated task performer to utilize cognitive

and physical resources to communicate and perceive the appropriate information to enable task performance.

Consideration of the information available in the task relationship diagrams when performing task allocation permits the designer to understand and potentially reduce the volume or complexity of information exchange between a human and machine. This tool may also help to reduce unwanted redundancy between the functions the human and the machine perform by clarifying the form of the information necessary to facilitate human decision making.

The function-to-task model requires the designer to identify the functions that are necessary to achieve the goals of the system and decompose these functions into leaf-level or atomic functions. The dependencies among these functions are then explicitly captured in a function relationship diagram. The functions are then allocated to an appropriate entity to form the basis of a task relationship diagram. Information flow between independent entities is then defined, identifying inherent tasks that are present in the allocation to provide communication. The form of the task relationship diagram is then evolved through application of five analysis tools to identify points in the TRD where adaptive automation could be easily accommodated. The application of this process model thus results in the allocation of tasks to the human, machine, or dynamically to the two entities.

The TRD resulting from a systematic implementation of the function-to-task design process model allows the designer to identify locations within a system where adaptive automation could provide benefit. However, this process model does not result in the design of an adaptive automation system but aids the designer in during the conceptual design portion of the user-centered design process (Gulliksen et al., 2003). Coupling the information from the task relationship diagram with the existing adaptive automation taxonomy, the designer can more effectively create well-targeted adaptive automation systems. Further, the tasks derived from this process can be used as input to existing interface design models, such as DIANE+ (Tarby and Barthet, 1996) or ConcurTaskTrees (CTT) (Paternò et al., 1997), which enable detailed design of the user interface.

Acknowledgments

The authors gratefully acknowledge the support of the Air Force Office of Scientific Research, which partially funded this research. The authors would also like to thank Maj Kennard Laviers, PhD, for his contribution of the Space Navigator game design. Finally, we would like to thank our anonymous reviewers who have provided helpful comments throughout the review process.

References

Annett, J., 2003. Hierarchical task analysis. Handbook of cognitive task design, 17–35.

Annett, J., Duncan, K. D., 1967. Task analysis and training design.

Blanchard, B. S., Fabrycky, W. J., 2006. Systems engineering and analysis.

Buede, D. M., 2011. The engineering design of systems: models and methods. Vol. 55. John Wiley & Sons.

Bye, A., Hollnagel, E., Brendeford, T. S., 1999. Human–machine function allocation: a functional modelling approach. Reliability Engineering & System Safety 64 (2), 291–300.

Byrne, E. A., Parasuraman, R., 1996. Psychophysiology and adaptive automation. Biological Psychology 42 (3), 249–268.

Calleja, J. A. B., Troost, J., 2005. A fuzzy expert system for task distribution in teams under unbalanced workload conditions. In: International Conference on Intelligent Agents, Web Technologies and Internet Commerce. Vol. 1. IEEE, pp. 549–556.

Card, S. K., Moran, T. P., Newell, A., 1986. The psychology of human-computer interaction. CRC Press.

Cheng-Leong, A., Li Pheng, K., Keng Leng, G. R., 1999. Idef*: a comprehensive modelling methodology for the development of manufacturing enterprise systems. International Journal of Production Research 37 (17), 3839–3858.

Delligatti, L., 2013. SysML Distilled: A Brief Guide To The Systems Modeling Language. Addison-Wesley.

Dorneich, M. C., Ververs, P. M., Mathan, S., Whitlow, S., Hayes, C. C., 2012. Considering etiquette in the design of an adaptive system. Journal of Cognitive Engineering and Decision Making 6 (2), 243–265.

Dzindolet, M. T., Peterson, S. A., Pomranky, R. A., Pierce, L. G., Beck, H. P., 2003. The role of trust in automation reliance. International Journal of Human-Computer Studies 58 (6), 697–718.

Endsley, M. R., 1999. Level of automation effects on performance, situation awareness and workload in a dynamic control task. Ergonomics 42 (3), 462–492.

Feigh, K. M., Dorneich, M. C., Hayes, C. C., 2012. Toward a characterization of adaptive systems: A framework for researchers and system designers. Human Factors: The Journal of the Human Factors and Ergonomics Society.

Fitts, P., Chapanis, A., Frick, F., Garner, W., Gebhard, J., Grether, W., Henneman, R., Kappauf, W., Newman, E., Williams, Jr., A., 1951. Human engineering for an effective air-navigation and traffic-control system. Tech. rep., National Research Council Committe on Aviation Psychology, Washington, D.C.

Geiselman, E. E., Johnson, C. M., Buck, D. R., 2013. Flight deck automation invaluable collaborator or insidious enabler? Ergonomics in Design: The Quarterly of Human Factors Applications 21 (3), 22–26.

Giese, M., Mistrzyk, T., Pfau, A., Szwillus, G., von Detten, M., 2008. Amboss: A task modeling approach for safety-critical systems. In: Engineering Interactive Systems. Springer, pp. 98–109.

Gulliksen, J., Göransson, B., Boivie, I., Blomkvist, S., Persson, J., Cajander, Å., 2003. Key principles for user-centred systems design. Behaviour and Information Technology 22 (6), 397–409.

Haarmann, A., Boucsein, W., Schaefer, F., 2009. Combining electrodermal responses and cardiovascular measures for probing adaptive automation during simulated flight. Applied Ergonomics 40 (6), 1026–1040.

Hollnagel, E., 2012. Fram: the functional resonance analysis method: modelling complex socio-technical systems. Ashgate Publishing, Ltd.

Itoh, M., 2011. A model of trust in automation: Why humans over-trust. In: SICE Annual Conference (SICE), 2011 Proceedings of. IEEE, pp. 198–201.

Itoh, M., Abe, G., Tanaka, K., 1999. Trust in and use of automation: their dependence on occurrence patterns of malfunctions. In: IEEE International Conference on Systems, Man, and Cybernetics. Vol. 3. IEEE, pp. 715–720.

Johnson, P., Johnson, H., Waddington, R., Shouls, A., 1988. Task-related knowledge structures: analysis, modelling and application. In: BCS HCI. Citeseer, pp. 35–62.

Jordan, D., Evdemon, J., Alves, A., Arkin, A., Askary, S., Barreto, C., Bloch, B., Curbera, F., Ford, M., Goland, Y., Guzar, A., Kartha, N., Liu, C. K., Khalaf, R., Knig, D., Marin, M., Mehta, V., Thatte, S., van der Rijn, D., Yendluri, P., Yiu, A. (Eds.), April 2007. Web Services Business Process Execution Language Version 2.0, 2nd Edition.

Kaber, D. B., Riley, J. M., Tan, K.-W., Endsley, M. R., 2001. On the design of adaptive automation for complex systems. International Journal of Cognitive Ergonomics 5 (1), 37–57.

Kidwell, B., Calhoun, G. L., Ruff, H. A., Parasuraman, R., 2012. Adaptable and adaptive automation for supervisory control of multiple autonomous vehicles. In: Proceedings of the Human Factors and Ergonomics Society Annual Meeting. Vol. 56. SAGE Publications, pp. 428–432.

Kirwan, B., 2005. Developing human informed automation in air traffic management. Contemporary Issues In Human Factors And Aviation Safety, 247.

Larman, C., 2004. Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development. Pearson Education, Inc.

Lee, J. D., See, K. A., 2004. Trust in automation: Designing for appropriate reliance. Human Factors: The Journal of the Human Factors and Ergonomics Society 46 (1), 50–80.

Lu, S., Paris, C., Linden, K. V., Colineau, N., 2003. Generating uml diagrams from task models. In: Proceedings of the 4th Annual Conference of the ACM Special Interest Group on Computer-Human Interaction. ACM, pp. 9–14.

MacDonald, W., 2003. The impact of job demands and workload on stress and fatigue. Australian Psychologist 38 (2), 102–117.

Manzey, D., Reichenbach, J., Onnasch, L., 2012. Human performance consequences of automated decision aids the impact of degree of automation and system experience. Journal of Cognitive Engineering and Decision Making 6 (1), 57–87.

Martinie, C., Palanque, P., Barboni, E., Ragosta, M., 2011. Task-model based assessment of automation levels: application to space ground segments. In: Systems, Man, and Cybernetics (SMC), 2011 IEEE International Conference on. IEEE, pp. 3267–3273.

Martinie, C., Palanque, P., Ragosta, M., Fahssi, R., 2013. Extending procedural task models by systematic explicit integration of objects, knowledge and information. In: Proceedings of the 31st European Conference on Cognitive Ergonomics. ACM, p. 23.

Merritt, S. M., Heimbaugh, H., LaChapell, J., Lee, D., 2012. I trust it, but i dont know why effects of implicit attitudes toward automation on trust in an automated system. Human Factors: The Journal of the Human Factors and Ergonomics Society.

Nakazawa, H., 1993. Alternative human role in manufacturing. AI & society 7 (2), 151–156.

Norman, D. A., Draper, S. W., 1986. User Centered System Design; New Perspectives on Human-Computer Interaction. L. Erlbaum Associates Inc., Hillsdale, NJ, USA.

Parasuraman, R., Cosenzo, K. A., De Visser, E., 2009. Adaptive automation for human supervision of multiple uninhabited vehicles: Effects on change detection, situation awareness, and mental workload. Military Psychology 21 (2), 270–297.

Parasuraman, R., Sheridan, T. B., Wickens, C. D., 2000. A model for types and levels of human interaction with automation. Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on 30 (3), 286–297.

Parasuraman, R., Sheridan, T. B., Wickens, C. D., 2008. Situation awareness, mental workload, and trust in automation: Viable, empirically supported cognitive engineering constructs. Journal of Cognitive Engineering and Decision Making 2 (2), 140–160.

Parasuraman, R., Wickens, C. D., 2008. Humans: Still vital after all these years of automation. Human Factors: The Journal of the Human Factors and Ergonomics Society 50 (3), 511–520.

Paterno, F., 2001. Towards a uml for interactive systems. In: Engineering for human-computer interaction. Springer, pp. 7–18.

Paternò, F., Mancini, C., Meniconi, S., 1997. Concurtasktrees: A diagrammatic notation for specifying task models. In: Human-Computer Interaction INTERACT97. Springer, pp. 362–369.

Prevot, T., Homola, J., Mercer, J., 2008. Human-in-the-loop evaluation of ground-based automated separation assurance for nextgen. In: Congress of International Council of the Aeronautical Sciences Anchorage, Anchorage, AK.

Price, H. E., 1985. The allocation of functions in systems. Human Factors: The Journal of the Human Factors and Ergonomics Society 27 (1), 33–45.

Reid, G. B., Nygren, T., 1988. The subjective workload assessment technique: A scaling procedure for measuring mental workload. Human mental workload 185, 218.

Ross, D., Jan 1977. Structured analysis (sa): A language for communicating ideas. Software Engineering, IEEE Transactions on SE-3 (1), 16–34.

Rouse, W. B., 1977. Human-computer interaction in multitask situations. Systems, Man and Cybernetics, IEEE Transactions on 7 (5), 384–392.

Rouse, W. B., 1981. Human-computer interaction in the control of dynamic systems. ACM Computing Surveys (CSUR) 13 (1), 71–99.

Sheridan, T. B., Parasuraman, R., 2005. Human-automation interaction. Reviews of human factors and ergonomics 1 (1), 89–129.

Sheridan, T. B., Verplank, W. L., 1978. Human and computer control of undersea teleoperators. Tech. rep., DTIC Document.

Tarby, J.-C., Barthet, M.-F., 1996. The diane+ method. In: CADUI. Vol. 96. pp. 95–119.

Valente, A., Carpanzano, E., 2011. Development of multi-level adaptive control and scheduling solutions for shop-floor automation in reconfigurable manufacturing systems. CIRP Annals-Manufacturing Technology 60 (1), 449–452.

Valente, A., Carpanzano, E., Nassehi, A., Newman, S. T., 2010. A step compliant knowledge based schema to support shop-floor adaptive automation in dynamic manufacturing environments. CIRP Annals-Manufacturing Technology 59 (1), 441–444.

van der Aalst, W., ter Hofstede, A., 2005. Yawl: yet another workflow language. Information Systems 30 (4), 245 – 275.

Van Welie, M., Van Der Veer, G. C., 2003. Groupware task analysis. Handbook of cognitive task design, 447–476.

Vanderdonckt, J., Tarby, J.-C., Derycke, A., 1998. Using data flow diagrams for supporting task models. In: DSV-IS (2). pp. 1–16.

Wickens, C. D., 2008. Multiple resources and mental workload. Human Factors: The Journal of the Human Factors and Ergonomics Society 50 (3), 449–455.

Woods, D. D., Cook, R. I., 2006. Incidents–markers of resilience or brittleness. Resilience Engineering. Concepts and Precepts. Ashgate, Aldershot, UK.

Vitae

Jason Bindewald received his B.A. degree in Computer Science from Gettysburg College, Gettysburg, Pennsylvania, in 2005. He received his M.S. degree in Cyber Operations in 2011 and is currently a Ph.D. Student in the Department of Computer and Electrical Engineering at the the Air Force Institute of Technology. He is a member of the Tau Beta Pi engineering honor society. His research interests are in the areas of adaptive automation, artificial intelligence, games, and machine learning with emphasis on human-machine interaction.

Michael E. Miller (PhD Virginia Tech) is an Assistant Professor at the Air Force Institute of Technology. His research interests include human systems integration, modeling and measurement of human performance and human interface design. Prior to joining AFIT, Dr. Miller spent more than 15 years with Eastman Kodak Company as a systems and human factors engineer. Dr. Miller holds over 80 US Patents in digital imaging and OLED systems. He is a senior member of SID, a member of the Human Factors and Ergonomics Society, a senior member of the IIE, and a member of INCOSE.

Gilbert L. Peterson is an Associate Professor of Computer Science at the Air Force Institute of Technology, and Vice-Chair of the IFIP Working Group 11.9 Digital Forensics. Dr. Peterson received a BS degree in Architecture, and an M.S and Ph.D in Computer Science at the University of Texas at Arlington. He teaches and conducts research in autonomous systems, digital forensics, and statistical machine learning. His research has been sponsored by the NSF, DARPA, AFOSR, AFRL, and JIEDDO. He has over 75 peer reviewed publication, and a book. In 2008, he received the Air Force Junior Scientist of the Year Category I award.