Proceedings of the IEEE International Conference on Information Reuse and Integration, pp. 351 – 356, Nov. 2004.

Coordinating Human Operators and Computer Agents for Recovery-Oriented Computing

Sreekanth K. Bhaskaran and Baback Izadi
Dept. of Electrical and Computer Engineering
State University of New York
New Paltz, New York 12561, USA

Lisa Spainhower
Systems and Technology Group
IBM
Poughkeepsie, New York 12601, USA

Abstract

This paper examines the errors committed by human operators of large networks and systems. It proposes a formal procedure in which system defense mechanisms are used to improve the coordination between human operators and computer agents. Further, it discusses and compares the effectiveness of different types of system defense mechanisms by performing experiments with web-based GUI screens. In the process, the paper offers definitions of human errors and proposes methods to quantify such errors. Our experimental results have shown that more layers of system defense can play a pivotal role in minimizing commonly encountered human errors.

1. Introduction

The study of service outages in large systems and networks points to human errors as the primary reason for the outage in a majority of the cases [1, 2, 3, 4]. Analysis of network outages consistently shows that operator errors account for more than 50% of unplanned downtime [1].

Operator error refers to an action performed by a human operator that is determined to be the root cause of a service disruption. Several classifications for human errors are found in [5, 6]. Very broadly, operator error can result from either a formulated procedure performed incorrectly by the operator or the procedure itself being incorrect in the first place [1]. In this paper, we examine the former. Other researchers have examined the building of user interfaces based on cognitive engineering methods to reduce human errors [7]. Human operators and computer agents could perform many of the tasks jointly. Partial Global Planning (PGP) is a coordination mechanism between humans and agents [8] that attempts to optimize the human and computer resources to achieve a common goal. PGP offers a dynamic task-sharing procedure for multi-agent systems that maximizes the capacity of every agent. Task allocation among several computer agents in real-time multiprocessor systems has been studied in detail. An optimal task allocation mechanism for real-time computing systems has been presented in [9]. However, the task allocation between computers and humans poses a different kind of challenge and is often dynamic. A direct effort to reduce the identified risks can bring about a net increase in the very same category of risks [10, 11]. There is a need to develop an analytical methodology that can perform task allocation adaptively [7]. Machine learning could be used effectively for this purpose and several machine-learning techniques have been developed, such as reinforcement learning [13], Q-based learning [14], and case-based reasoning [15].

This paper examines non-deliberate operational faults made during human-computer interaction and presents a formal procedure to reduce human errors. It proposes to use machine-learning methods employing case-based reasoning to reduce human errors. The rest of the paper is organized as follows. The next section introduces the concept of system defense. Section 3 discusses how machine learning can be used effectively for improved system defensiveness thus achieving better coordination between humans and computer agents. Section 4 presents a formal procedure in which system defense mechanisms are used to improve the coordination between human operators and computer agents. Section 5 discusses our experimental study that analyzes the effect of machine learning on human and computer coordination. Finally, the concluding remarks are given in Section 6.

2. System Defense

It is virtually impossible to eliminate human operator intervention to achieve the goals of large systems. Figure 1 shows a sequence that could be visualized as the interleaving of operator actions and agent actions. After every operator action, the agent could examine current operator behavior against past actions. Moreover, it could generate warnings for the user if the operator action is a deviation from successful actions taken by operators in the past.

We define system defensiveness as the intelligence built into the system to warn the users of potentially detrimental actions. It could be considered as a method of implementing the automated agent checks. The system

reacts to operator actions with a defensiveness warning and does not let the operator proceed unless an acknowledgement is provided to the defensiveness warning.

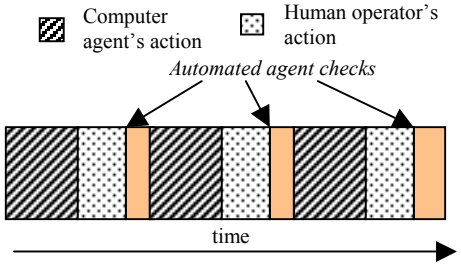

Figure 1. Agent checks that could reduce operator error

The system determines the level of attention demanded from the operator, as illustrated in Table 1. In the simplest form, the operator action severity is classified as of having high, medium or low impact to the revenues of the company. Accordingly, the system could demand acknowledgement from the operator in varying degrees. For instance, there could be a simple user confirmation for a low impact action, an acknowledgement requiring operator password for an action of medium impact and confirmation from multiple operators for an action with high impact. Diversity among operators [5] and distribution of trust [10] could be very effective safeguards against common operator errors.

Table 1. System Defense strategies

Actions	*Weights (Cost in $)*	*Defense Strategy*
Action 1	Low	Simple user confirmation
Action 2	Medium	User Confirmation with password
Action 3	High	Confirmation from multiple users

System defense can be static or dynamic, preemptive or informatory. A static system defense mechanism would pose the same defense question to the user every time the user performs a specific action. An example of this would be file deletion in Microsoft Windows; when a user attempts to delete a file, the system asks the user to acknowledge that the file deletion is intentional. Such defense strategy is sufficient for most applications. A deficiency with the scheme is that people tend to increasingly ignore it if the same defense question is posed over and over again. The result is usually a delay in the execution of the action (two 'OK' clicks instead of one) but not a more careful execution.

A dynamic defense strategy would incorporate the current environmental conditions and past knowledge before posing a defense question. Every time the user initiates an action, the intelligent agent gathers environmental data, searches the database containing past actions, and poses a defense question that is custom-made for the specific situation. Machine-learning agents, capable of adapting to the environment and even learning from other agents, can be effective towards achieving autonomic computing [12]. Of course, the downside is the computational overhead and the delay associated with it. The overhead could be justified in critical systems and networks, which could reduce operator errors by a more careful coordination between the humans and computer agents. The defense question and information can be purely informative, which gives the user a chance to correct any inadvertent mistakes. When the defense is preemptive, the information is not only informatory but the agent will not proceed with the execution of the user action unless the user overrides system defensiveness.

3. Machine Learning in System Defensiveness

Machine learning can be used effectively to pose defense questions that are customized for each instance of user action. To do so requires maintaining a record of past actions, classifying them and prompting the user in real-time with a defense question that is based on the knowledge thus gathered. Knowledge-based systems could also be used very effectively to perform sensible system defense actions. Given that the intervention of a human operator is inevitable in the operation of large networks and systems, and that human error is inevitable, the agent's job is to pose sensible defense questions to the user and avert as many undesirable actions as possible. A dynamic defense mechanism that gets its inputs from a variety of sources that could include other agents, the human operator and the knowledge of past events could prove effective in reducing many of the commonly encountered operator errors.

3.1 Case-based Reasoning

We found case-based reasoning [15] particularly attractive for machine learning of tasks because of its simplicity. It was chosen for building the agent discussed later. The agent stores past successful tasks in a database and a matching is performed between the task in question and other instances of the same task stored in the database.

A task consists of generic sub-tasks that could be considered as an ordered pair of attributes and values. An attribute refers to an action. The value of an attribute is a parameter that qualifies the attribute in specific instance of occurrence of the attribute.

$T_i = \{S_{i1}, S_{i2}, \ldots\ldots, S_{im}\} = \{(A_{i1}, V_{i1}), \ldots\ldots(A_{im}, V_{im})\}$

To illustrate this, let us consider two tasks with their attributes and values, as shown in Figure 2. *Task 1* is to make an omelet and *Task 2* is to make pancakes. In Figure 2, 'Utensil placed on fire' is an attribute. The value 'pan' qualifies this attribute further for the current instance. Similarity of two attributes, *x* and *y* is defined by *Sim-a(x, y)* in the simplest form as

$$Sim\text{-}a\ (x,y) = \begin{cases} 1 & if \quad x = y \\ 0 & otherwise \end{cases} \qquad (1)$$

Similarity of values could be defined in the same way.

$$Sim\text{-}v\ (x,y) = \begin{cases} 1 & if \quad x = y \\ 0 & otherwise \end{cases}$$

In the example shown in Figure 2, the first attribute of both tasks is identical. Therefore, $Sim\text{-}a\ (A_{i1}, A_{j1},) = 1$. Distance between two sub-tasks S_{ir} and S_{js} is defined by

$$Dist\ (S_{ir}, S_{js}) = Sim\text{-}a\ (A_{ir}, A_{js}) \wedge Sim\text{-}v\ (V_{ir}, V_{js}) \qquad (2)$$

In the example in Figure 2, the attributes as well as value for the first item are the same. Therefore, $Dist\ (A_{i1}, A_{j1},) = 1\wedge\ 1 = 1$. Similarity between tasks could be computed using the distance between every pair of sub-tasks.

$$Sim\ (T_i, T_j) = \sum_{r=1}^{m} \sum_{s=1}^{n} Dist\ (S_{ir}, S_{js}) \qquad (3)$$

In the example in Figure 2, the similarity between tasks is

$Sim\ (T_i, T_j) = (1 + 1 + 0 + 1 + 1 + 1) = 5.$

A demand on the similarity of tasks T_i and T_j is defined by Equation 4, where α indicates a pre-specified threshold value.

$$Dem\text{-}Sim\ (T_i, T_j) = \{\ Sim\ (T_i, T_j) > \alpha\ \} \qquad (4)$$

There are several issues with having such a simplistic analysis of tasks. For one, the attributes and their values in the tasks alone do not determine the extent of similarity between the tasks. For instance, in the previous example, even if the sequence of sub-tasks in Task 2 were totally reversed, the same similarity would be computed. This is an anomaly because in the real world, the sequence of actions in a task determines, to a great degree, the level of similarity between tasks. Taking this into account, Equation 3 is modified as

$$Sim\ (T_i, T_j) = \sum_{r=1}^{m} \sum_{s=1}^{n} (Dist\ (S_{ir}, S_{js})) + SF\ (T_i, T_j)$$

	Task 1: Making Omelet	
Steps	**Attribute**	**Value**
1	Utensil -- placed on fire	Pan
2	Wait for ---	1 min
3	Add --- in utensil	Broken eggs
4	Wait for --	0.5 min
5	Flip item --	Once
6	Serve in ---	Plate

Task 2: Making Pancakes	
Attribute	**Value**
Utensil -- placed on fire	Pan
Wait for ---	1 min
Add --- in utensil	Pancake batter
Wait for ---	0.5 min
Flip item ---	Once
Serve in ---	Plate

Figure 2. Calculating similarity of tasks

where *SF* is the Sequence Factor and is given by

$$SF\ (T_i, T_j) = \sum_{k=1}^{m} \left(\sum_{r=1}^{m} \sum_{s=1}^{n} Sim\ (A_{ikr}, A_{jks}) + \sum_{r=1}^{m} \sum_{s=1}^{n} Sim\ (V_{ikr}., V_{jks}) \right) \qquad (5)$$

To compute *SF* between two tasks T_i and T_j, matching sequences are located starting from the first attribute. In Equation 5, the variable *k* denotes the iteration. In the example in Figure 2, for the first iteration *(k = 1)*, we start at the first sub-task of Task T_1 and T_2 and see that the attribute and value are matching. i.e. $Sim\ (A_{111}, A_{211}) = 1$ and $Sim\ (V_{111}, V_{211}) = 1$. This contributes a value of 1 + 1 = 2 to *SF*. Next, we increment *r* and s and still see that the attribute and value match for the second item as well. Again $Sim\ (A_{112}, A_{212}) + Sim\ (V_{111}, V_{211}) = 1+ 1 = 2$ is contributed to *SF*. In the third item, the attribute matches but not the value. This adds 1+ 0 = 1 to *SF*. Thus, at the end of iteration 1, *SF* gets a value of 11. We start the second iteration from the attribute 'Wait for?' starting from $Sim\ (A_{121}, A_{221}) + Sim\ (V_{121}, V_{221}) = 1+ 1 = 2$ and proceed in a similar way to get a value of 9. At the end of sixth iteration, *SF* will have a value of 11+9+8+6+4+2 = 40.

The sequence factor could have the highest possible value if the compared tasks, T_i, T_j are identical. In the example given in Figure 2, there are six sub-tasks and their maximum *SF* value is 12+10+8+6+4+2 = 2(6+5+4+3+2+1) = (2× 6× 7)/2 = 42. In general, the

maximum value of *SF*, for two identical tasks with n sub-tasks is n (n+1). The tasks discussed above are simple in that they are composed of sub-tasks, each having an attribute and a value. More complex tasks will have several levels of abstraction. The sub-task of a task could be so complex that the sub-task itself would need to be broken down into smaller sub-tasks. This is represented in Figure 3. The formulae discussed in equations (3) and (5) could be modified to have several levels of nesting.

Task	Sub-task level-n	…	Sub-task level-2	Sub-task level-1	Attributes, values
Increasing granularity →					

Figure 3. Decomposing a task.

3.2 Quantifying Human Errors

Quantification of human errors is necessary to analyze the kind of mistakes operators commit and to drive the creation of automation benchmarks. Measuring human errors is challenging, especially because errors and tasks vary across applications. The definition of what an error is would be different for each kind of task. For an installer of LAN, connecting incompatible devices would be an error and for an administrator who uses GUI for provisioning services, a typo would be an error. We concentrate on unintentional GUI operator errors. The GUI is also more suitable for system defensiveness because humans better observe it.

Human error is quantified based on the following criteria:

- The extent of damage that it creates in terms of revenues, E_r.
- The extent of variation from the expected action, E_v.
- The amount of delay it creates; this includes undoing any harm caused by the erroneous action plus the time required to perform the correct action, E_d.

The extent of damage caused in terms of revenues is subjective and would vary from place to place and time to time. The benefit-loss function (BLF) discussed in [15] attempts to quantify errors of this kind. In that, a system engineer assigns the accuracy loss in an event to a consequent benefit loss value. Error could also be expressed as a function of the three measures described above.

$$E = f(E_r, E_v, E_d) \tag{6}$$

The proposed error quantification that is based on variation from an expected action (E_v) could be calculated from the principles of similarity. Maximum similarity of two tasks is $Sim(T_i, T_j)$ in Equation 5 could be normalized to have a maximum value of 1. Therefore, dissimilarity between tasks is defined as,

$$Dissim(T_i, T_j) = 1 - Sim(T_i, T_j) \tag{7}$$

Dissimilarity between erroneous instance and last successful instance could be used as a measure of error.

$$E_v = Dissim(X_i, X_i) = 1 - Sim(X_i, X_i) \tag{8}$$

where X_j is erroneous task instance and X_i is successful task instance.

4. A Formal Procedure

Systems that try to improve human-computer coordination could use the following step-by-step procedure to achieve a dynamic defense mechanism.

i. Identify the operation for which dynamic system defense could be applied. This might appear trivial but it is crucial that the selected operation satisfies some basic criteria like having a logical beginning and end.
ii. Analyze the operation and create valid task paths. The operation is split into several sub-tasks. The set of valid task paths is determined.
iii. Build the infrastructure. This is largely a software architectural issue. The choice of infrastructure depends on the size of the desired database and the technology used in the operation under study.
iv. Create the learning system. The learning system is created by inputting seed cases for all valid paths into the database.
v. Periodically maintain the learning system. Failures do occur and when such occurrences are analyzed, the entries in the database pertaining to the failures need to be removed. This is because the agent should learn from good examples and not from bad examples. Seed cases of new tasks might have to be added to the database.
vi. Fine-tune and customize. The behavior varies across different people and their fatigue levels. Individual characteristics of operators as well as general human behavioral characteristics could provide vital clues regarding psychological and physical fatigue level. These could in turn be used for appropriate defense question.
vii. Analyze and specify the performance of the defense mechanism. It is important to determine whether the learning agent put in place is actually beneficial in reducing the human operator errors. Another important consideration would be to see whether the delay and computational overhead associated with the

defense mechanism could be justified by its benefits. The delays should be minimized using optimal queries to the database and by keeping the database size manageable. The upper limit for database size could be determined using the acceptable delay and its lower limit could be determined by the least acceptable level of learning deemed necessary .

5. Experiments and Results

To study the effect of several layers of system defense on unintentional human operator errors, a series of experiments were devised and conducted. The experiments studied the effectiveness of agents in using machine learning to reduce human operators.

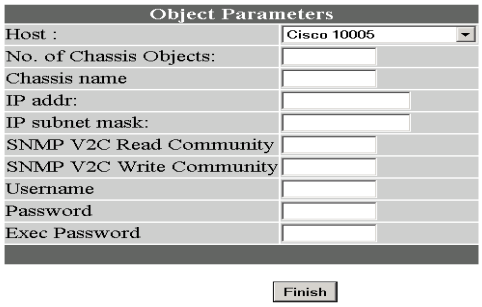

Figure 4. GUI used for experiments

A GUI screen similar to the one of the most popular commercial IP products (Figure 4) was chosen for the experiments. A large group of students with sufficient computer skills were asked to enter five sets of data in the GUI screen from a printout. For the first experiment, the values entered by the user were committed to a mySQL database without any defense mechanism. In the second experiment, called Static defense, the entered values were displayed back to the user and user confirmation was requested before committing them into the database. The user responses for each of the experiments were recorded in a database. The users' responses and the number of human errors encountered were analyzed. Figure 5 shows the percentage of fully successful entries for the experiments. Accordingly, our Static defense mechanism improved the system performance, compared with no defense mechanism, by nearly 9%. In the third experiment, called Dynamic-1, the agent software performed a matching of the received values with the ones obtained from the past using PHP scripts, and displayed the matched values along with the entered values back to the user. The data was only accepted to the database after the user decided that the extent of displayed match was acceptable. Surprisingly, Dynamic-1 resulted in worse performance than the one with even no defense mechanism. In the final experiment, called Dynamic-2, the matched values along with the entered values were displayed to the user and additional inspection was requested only if the mismatched value crossed a predetermined threshold. The threshold level was determined by taking into account that there were five sets of data and there could be several incorrect entries. Our experimental result shows that Dynamic-2 defense mechanism used in the fourth experiment resulted in a smaller percentage of errors compared to the other three experiments; it resulted in over 90% success rate. The defense mechanism in the third experiment offered no improvement to the case where there was no system defense at all. Possibly, more information resulted in more confusion for the operators. But when the user was interrupted with the additional information only when it was necessary, that is, only when a certain threshold was crossed, there were fewer errors committed. This strengthens the idea that system defense mechanisms could be effective in reducing human operator errors when relevant information is provided at the proper time.

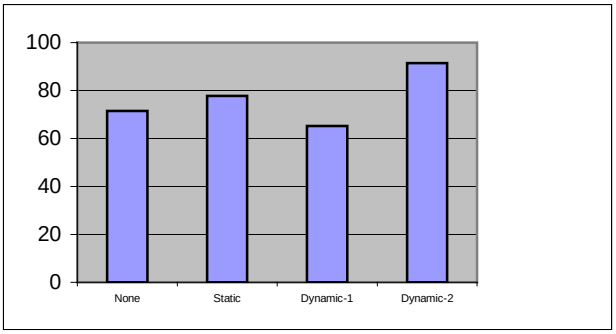

Figure 5. Percentage of successful entries for experiments

The types of errors committed in each of the experiments were also analyzed. The errors were classified into two broad categories. Errors of omission [5], where an element of a sequence <a, b, c, d> is accidentally left out as in <a, b, d>, and errors of commission, where another element is placed instead of the right one, e.g., <a, x, c, d> instead of <a, b, c, d>. The error classification for each of the three experiments is shown in Figure 6.

It was observed that the percentage of total errors due to omission was reduced significantly when a static defense mechanism was used. Most of the errors in Dynamic-1 were of omission type. Finally, Dynamic-2 resulted in the least omission and commission errors.

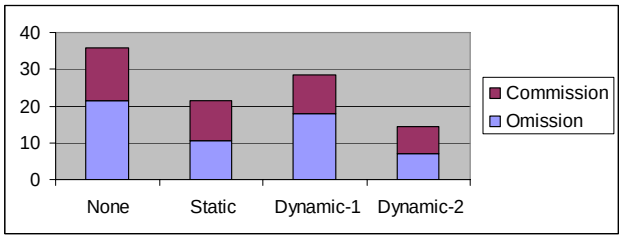

Figure 6. Percentage of errors observed for various defense mechanisms

6. Conclusion

In this paper, we examined a mechanism to reduce the errors committed by human operators of large networks and systems. A formal procedure in which system defense mechanisms are used to improve the coordination between human operators and computer agents was presented. The paper provided methods to analyze and quantify the errors committed by human operators. It studied and compared human errors in scenarios with different levels of system defense. A simple static defense mechanism was determined to be very effective against interface errors committed by operators. A defense mechanism that employed the past knowledge of user entries and interrupted the user only when there was considerable mismatch resulted in minimum operator errors.

A dynamic defense strategy employing more criteria would likely have better results. These criteria could include the state of the operator [13], which is a function of the operator's fatigue level and cognitive capability level. For instance, the layers of defense could adapt to become more restrictive as the fatigue level increases. However, this could bring about the exact opposite effect and is best studied with further experiments. Further research will compare and study intelligent agents to accurately determine the benefits of dynamic defense.

References

1. D. A. Patterson, D. Oppenheimer, "Architecture and Dependability of Large-Scale Internet Services," *IEEE Internet Computing*, pp. 41-49, Sep-Oct 2002.
2. D. A. Patterson, A. Brown, P. Broadwell, G. Candea, M. Chen, J. Cutler, P. Enriquez, A. Fox, E. Kiciman, M. Merzbacher, D. Oppenheimer, N. Sastry, W. Tetzlaff, J. Traupman and N. Treuhaft, "Recovery-Oriented Computing (ROC): Motivation, Definition, Techniques, and Case Studies," *UC Berkeley Computer Science Technical Report UCB//CSD-02-1175*, March 15, 2002.
3. J. M. Christensen and J. M. Howard, "Field Experience in Maintenance. Human Detection and Diagnosis of System Failures," *Proceedings of the NATO Symposium on Human Detection and Diagnosis of System Failures,* J. Rasmussen and W. Rouse (Eds.). New York: Plenum Press, pp. 111–133, 1981.
4. J. Gray, "Why Do Computers Stop and What Can Be Done About It?" *Symposium on Reliability in Distributed Software and Database Systems*, pp. 3–12, 1986.
5. Y. DesWarte, K. Kanoun, J-C Laprie, "Diversity against accidental and deliberate Faults*." IEEE Computer Security, Dependability, and Assurance: From Needs to Solutions*," pp. 171-182, July 1998.
6. R. E. Fields, P. C. Wright and M. D. Harrison, "A Task Centered Approach to Analyzing Human Error Tolerance Requirements.," *Proc. Of the Second IEEE International Symposium on Requirements Engineering (RE'95),* pp. 18-26, 1995.
7. W. P. Marshak, M. M. Pohlenz, "Modeling for Cognitive Engineering of User Interfaces: Need, Basis and One Promising Implementation," *Proc. of the 3rd IEEE Symposium on Human Interaction with Complex Systems (HICS'96),* pp. 170-178, 1996.
8. K. Decker and V. Lesser, "Designing a family of coordination mechanisms," *Proceedings of the First International Conf. On Multi-Agent Systems (ICMAS-95)*, pp. 73-80, June 1995.
9. K. H. Kim, Y. Kim, "An Experimental Investigation of the Potential of BLF-driven Scheduling of Real-Time Threads," *Proc of the 2nd IEEE International Conference on Engineering of Complex Computer Systems (ICECCS'96)*, pp. 60-66, 1996.
10. D. Dalcher, "Trust, Systems and Accidents: Designing Complex Systems," *Proc. of the 10th IEEE International Conference and Workshop on the Engineering of Computer-Based Systems (ECBS'03),* pp. 31-39, 2003.
11. A. Wildavsky, "Searching for Safety," *Transaction Books, Oxford*, 1988.
12. J. O. Kephart, D. M. Chess, "The vision of Autonomic Computing," *IEEE Computer magazine,* pp. 41-50, January 2003.
13. L.P. Kaelbing, M. L. Littman, and A. W. Moore, "Reinforcement learning: A survey," *Journal of AI Research,* no.4, pp. 237-285, 1996.
14. C. J. Watkins, " Learning from Delayed Rewards," *PhD Thesis, King's College, Cambridge University,* 1989.
15. G. Weiss, "Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence", *Cygnus Software Ltd*, August 2000.