The Foundations of Information Push and Pull [1]

George Cybenko
gvc@dartmouth.edu
Brian Brewington
brew@dartmouth.edu

Thayer School of Engineering
Dartmouth College
Hanover, NH 03755 USA

Abstract

Information *push* and information *pull* have recently emerged as useful concepts to describe the operation of distributed information resources. Information push, in particular, is becoming closely associated with intelligent agent functionality. Loosely speaking, if a user requests and receives a very specific piece of information, this is information pull. If information is sent in anticipation of the user's need, or the agent's response includes information not directly solicited, then the situation is characterized as information push. Intuitively, junk mail (electronic or paper), television newscasts and wirefeeds are examples of information push. New web services such as Pointcast and Informant are examples of more selective push technologies. Web browsing, library searches, and telephone white pages are traditional examples of information pull. Clearly, these categorizations can be ambiguous and are easily lost in semantics. The main goal of this paper is to formalize these concepts and describe a mathematical framework around which further work can be more precise. Specifically, we develop a stochastic framework based on Markov models to describe an ambient environment and an agent system. Depending on the relationships between the environment, the agent and the user's performance criterion, a continuum of possible information push and pull scenarios can be described. Some basic analytic results concerning the operation of a push/pull information system are derived.

1 Introduction

Interest in agent technology is burgeoning as the amount of raw information grows faster than a typical human user's ability to keep pace. In fact, the term *agent* [14, 8, 4, 11, 5, 12] is now so commonplace it is at risk of becoming synonymous with the concept of a computer "program." The purpose of this short paper is to introduce some quantitative models for agent-like behavior so that future discussions about agents can have a more solid foundation. In particular, we discuss the notions of information push and pull in some detail as well as their relationship to agent systems.

We define agents to be programs that perform relatively complex, semantic tasks autonomously. Further discussion and definitions can be found in the recent literature, for example [15]. Our interest is in information agents that act as intermediaries between users and raw information resources, either locally or remotely. Agents may or may not be computer programs but they essentially perform computational tasks with respect to processing information. Their role as intermediaries can be useful for several reasons:

[1] Partially supported by Air Force Office of Scientific Research grants F49620-95-1-0305 and AFOSR DOD Multidisciplinary University Research Initiative grant F49620-97-1-0382. To appear in *Mathematics of Information*, Institute for Mathematics and Applications Proceedings, 1997

- FILTERING – When a well-defined information source cannot be monitored by a user because of its size or update rate, it may be possible to develop an agent to act as an intermediary filter. The filtering operation takes the source information input and passes along to the user either a subset of the information or an abstraction of it. Newspapers are examples of traditional filtering agents by virtue of their editorial role – the editorial staff of a newspaper has access to wire services and reporters' stories from which a subset of information is passed along to the user. Another traditional example of filtering is a simple smoke detector device. The detector monitors the environment (typically the optical transmission properties of air) and communicates an abstraction of that environment to the user – no alarm signal when there is no smoke, an alarm signal when there is smoke (or another impediment to light transmission).
- LOCATING – Filtering agents operate on a constant source of information. By contrast, locating agents must actively probe the environment in search of information to satisfy the user's information requirements. Traditional examples of locating agents are web search engines such as Infoseek, Altavista, Lycos and Excite. Travel agents and good stock analysts are other examples exhibiting locating agent attributes.
- ORGANIZING – In addition to locating and filtering, agents can serve a user by organizing information semantically. A good example of organizing agent functionality from everyday life is performed by television meteorologists. A weather report represents a certain organization of raw meteorological information, typically involving a large degree of filtering with little locating and most importantly an abstraction and organization of information into an output that is semantically meaningful and concise. Several Web services, such as PointCast[2], organize information to suit individual user's requirements.
- ALERTING – Clipping services alert users by storing search criteria and managing *standing queries* based on those criteria. A web application we have developed, the Informant[3], combines filtering, locating and organizing in one application.

Of course, there are many more examples of agent functionality. Moreover, the above categories make intuitive sense but stand to benefit from the development of a more precise functional taxonomy. We believe that this will be an active and fruitful area of research in the coming years.

In this paper, we focus on specific abstractions that all agent systems appear to exhibit – there is an underlying, stochastically changing environment, the agent implicitly has a model of that environment as well as a model of what information the user wants from that environment, and the user has a performance model of the agent within that environment. Using these ingredients, we formalize and compare several concepts of information push and pull as well as introducing performance measures for agents.

In particular, we develop three quantifiable measures: $PUSH_s$, $PUSH_n$ and $COMPLEXITY$ so that every agent can be described by a triple of numbers, called the *push/pull metric*:

$$(PUSH_s, PUSH_n, COMPLEXITY_{AU})$$

where $PUSH_s$ is the ratio of messages, measured in byte counts, that go between a user and an agent; $PUSH_n$ is the ratio between the number of messages, irrespective of size, that go between a user and an agent; and $COMPLEXITY_{AU}$ is a measure of complexity reduction accomplished by an agent from a user's point of view. Our examples show that different push technologies can have quite different metrics. Finally, we present some analytic results about stochastic versus deterministic agents.

2 User, Agent and Environment Models

In order to develop the ideas of information push and pull, we will assume that the enviroment, agent and user all have a well-defined instantaneous *state* in the sense of systems theory, automata theory or physical

[2]See http://www.pointcast.com.

[3]See http://informant.dartmouth.edu. The Informant is a clipping service for the World Wide Web. It periodically runs web search engine queries based on a user's standing request.

mechanics. Loosely speaking, knowledge of the state is used to estimate the future states of the system but additional historical information about previous states does not improve that estimate[4].

Transitions between states can be either stochastic or deterministic. The environment, agent and user can communicate with each other using "messages." Those messages are like external forces (in mechanics) or controls (in the control theory sense). How these messages affect the state is governed by the system's dynamics.

To formalize the above discussion, let $\mathcal{S}_E, \mathcal{S}_A$ and $\mathcal{S}_U$ denote the environment, agent and user state spaces. In each case, there is an instantaneous state, $s_E(t) \in \mathcal{S}_E, s_A(t) \in \mathcal{S}_A$ and $s_U(t) \in \mathcal{S}_U$ at time t. Messages can be exchanged between each of these components – $a_{\alpha\beta}(t)$, where α and β are one of E, A and U, denotes a message being sent from component α to component β at time t. For example, $a_{UA}(t)$ is a message (information) sent from the user to the agent at time t. The distribution of future states of a system depends on the current state, the dynamics of the system and received messages during the intervening time interval. In this way, messaging is seen to be a type of external control on each system.

As outlined above, our push/pull information model consists of three ingredients: an environment, an agent and a user. In order to define each of these more precisely and the interrelationships between them, we need to review the concepts of controlled Markov processes and deterministic automata [2, 10].

A *controlled Markov process* consists of a state space, a set of actions and a set of transition probabilities. The transition probabilities between states are determined by the actions. More precisely, the state space is a set of states, $\mathcal{S} = \{s_i\}$. At time t, the state of the system is a random variable $s(t)$ which can assume values from the state space, $\mathcal{S}$. We write $s(t) = s_i \in \mathcal{S}$ to mean that at time t, the system state is s_i.

The actions, $\mathcal{C} = \{c_j\}$, control transition probabilities between states. At time t, the action applied is denoted by $c(t)$ which assumes values from $\mathcal{C}$. Not all actions can be applied to all states. However, when an action, c_j is applied to the system in state $s(t-1) = s_k$, the resulting new state is determined by the conditional probability

$$p(s_i, c_j, s_k) = Prob(s(t) = s_i | s(t-1) = s_k, c(t-1) = c_j).$$

Note that the probability is independent of time, t.

The Markov nature of the system captures the fact that any past history of the system is irrelevant to the transition probabilities – only the current state affects those probabilities. Formally, let the system's state at time t be the random variable $s(t)$ and the action applied at time t be $c(t)$. Then

$$\begin{aligned} Prob(s(t) = s_k | s(t-1) = s_{j_1}, c(t-1) = c_{i_1}, s(t-2) = s_{j_2}, c(t-2) = c_{i_2}, s(t-3) = c_{j_3}, ...) \\ = p(s_i, c_j, s_k) = Prob(s(t) = s_k | s(t-1) = s_{i_1}, c(t-1) = c_{j_1}). \end{aligned}$$

Most causal systems can be described in terms of a Markovian representation using some part of the history of the process as its state [13].

Time is discrete in our model, advancing in uniform increments. Actions are externally triggered events – that is, some external controlling mechanism dictates the timing and nature of the actions. That external action could be generated by another system, deterministic or stochastic.

A *deterministic automaton* consists of a set of states, $\mathcal{S} = \{s_i\}$, and actions $\mathcal{C} = \{c_j\}$ which deterministically move the system between states. Again, not all actions may be allowed in all states.

Whereas controlled Markov processes make stochastic transitions between states with probabilities determined by the actions, deterministic automata make nonrandom transitions. Deterministic automata are therefore a subset of controlled Markov processes in which the transition probabilities have values either 0

[4]In classical Newtonian mechanics, instantaneous position and momentum are state variables in the sense that the future evolution of the system can be determined from those state values. The ability to determine future states depends on being able to observe the instantaneous states and on knowledge of the dynamics governing state evolution. If either of these is less then perfect, predicting the evolution will not be precise. A financial market, for example, is difficult to model because it is hard to identify the correct notion of state and the market dynamics are poorly understood. In such a situation, a stochastic model is needed whereby the uncertainty about states and dynamics are handled by stochastic modeling.

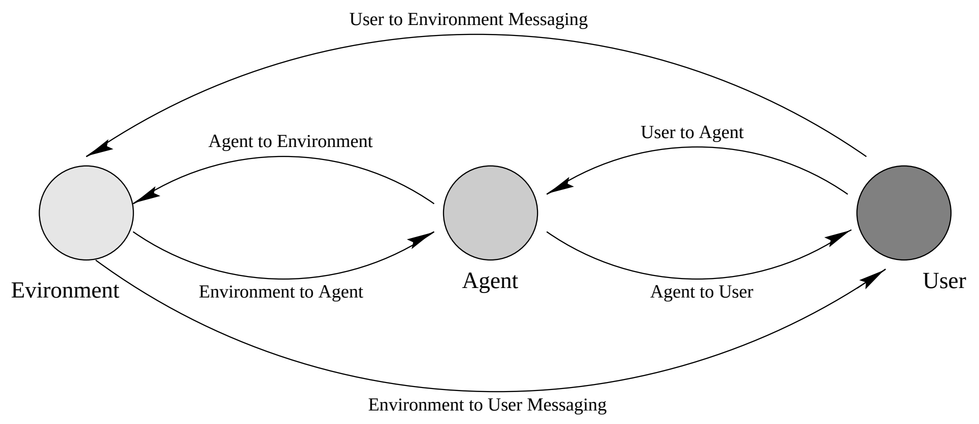

Figure 1: A General Framework for Expressing Interactions between Environments, Agents and Users

or 1. If an action, c_j in a deterministic automaton changes the system state from s_i to s_k then we can write

$$\begin{aligned} p(s, c_j, s_k) &= 1 \text{ if } s = s_i \\ &= 0 \text{ otherwise} \end{aligned}$$

which makes the controlled Markov process effectively deterministic. In light of this, deterministic automata are a subclass of controlled Markov processes. See [1, 2, 9] or other texts on stochastic control for more details and discussion.

As previously mentioned our push/pull information model consists of three ingredients: an environment, a user (observer) and an agent. We describe each separately next.

- *Environment* – The *environment* is the object of interest in most applications, and typically operates independently of the user and agent. That is, in many agent applications, the agent has only limited control over the environment, as it largely evolves according to its own dynamics. The environment is modeled as a controlled Markov chain, consisting of a state space, $\mathcal{S}_E = \{s_i^E\}$, action space $\mathcal{C}_E = \{c_i^E\}$, and transition probabilities: $\{p_E(s_i^E, c_j^E, s_k^E)\}$, being the probability of a transition from state s_i^E to state s_k^E under control c_j^E. In this way, the environment is seen to be a stochastic process possibly under the control of either the agent or the user. The controls are communicated as messages from either the user or agent. In many cases, the external control on the environment plays a minor role if any at all.

- *Agent* – An agent is modeled by a deterministic automaton, consisting of a state space $\mathcal{S}_A = \{s_i^A\}$, and actions $\mathcal{C}_E = \{c_j^E\}$ which determine state transitions – for an action, $c^E \in \mathcal{C}_E$, and a state $a^E \in \mathcal{A}_E$, if $c^E(a^E)$ is defined, it is the next state of the system. We model the agent deterministically because it is essentially a computer program. However, because the agent interacts with the environment which is a stochastic process, it can exhibit random behavior as discussed below. The agent's actions are internal state transitions as well as dependent on messages from the environment and user.

- *User* – The user is also modeled by a deterministic automaton with state space $\mathcal{S}_U = \{s_i^U\}$ and actions $\mathcal{C}_E = \{c_j^E\}$ which control state to state transitions. The user can interact with the environment and the agent through messaging.

Users, agents and the environment affect each others' states by sending messages. Special states are reserved for message sending. Figure 1 shows the general interrelationships between user, agent and environment and the possible message pathways.

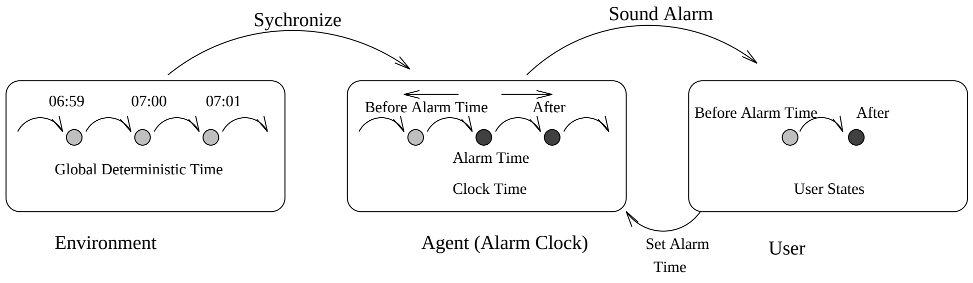

Figure 2: An Alarm Clock Agent

3 Examples

In this section, we briefly describe several push/pull information scenarios as examples of the above framework.

3.1 An Alarm Clock

An alarm clock is perhaps the simplest example of an agent, albeit not typically thought of as a computer-based agent technology. Nonetheless, it serves to illustrate the basic ideas in an elementary manner.

The underlying environment is that of *global time*, deterministically advancing in the common sense of time. The alarm clock is an agent with a mechanical or electrical simulation of that global time which monitors whether the estimated global time is before or after the set alarm time. If the time is after the alarm time, the alarm sounds alerting the user. The user only cares if the time is the set alarm time or not.

In this example, synchronization between the clock's time and global time is done rarely as is the user's setting of the alarm time. Figure 2 illustrates the operation of an alarm clock in the general framework of Figure 1.

Notice that the amount of communication passing between the environment and the agent (alarm clock) is small but the agent has a huge state space (time) and abstracts that into two states for the user - before and after alarm time. We will explore this aspect further in a later section.

3.2 A Smoke Alarm

A smoke alarm abstracts its environment by constantly monitoring some aspect of it that is highly correlated with danger to the user. As such, the smoke alarm is a metaphor for many different types of technologies such as safety systems in power plants, medical monitoring devices in intensive care units and automobile "idiot" lights. In these situations, the environment is treated stochastically; were it purely deterministic, it would in principle be possible to track its evolution once and for all after one of its states has been determined by the user. Whether the stochastic nature is due to intrinsic randomness or due to modeling uncertainty is irrelevant.

The true state space of the environment is large and complex – basically mirroring the state space of the world. A small subset of the environmental state space corresponds to "fire" conditions and most but not all of those situations create smoke that triggers the alarm. Burning toast causes smoke which triggers the alarm but is not in general considered a dangerous condition. Conversely, there are conditions (steam, dust, etc.) which trigger the alarm but do not correspond to smoke or signify danger.

The alarm is in one of two states – triggered or not. From the user's point of view, there are two error modes. The alarm can go off when no danger exists (false alarm situation) or a dangerous situation exists but no alarm is sounded (missed detection). Similarly, the user can be modeled as being in one of two states – safe or in danger. Those two states do not exactly correspond to the alarm being sounded but are highly correlated. We discuss the relationship of the error modes and Type I and Type II errors in hypothesis testing in a later section.

Figure 3 depicts the operation of a smoke detector in our general framework. Note that while the amount of information passing between the environment and the alarm is huge, the state space of the agent (alarm) is small with the net result that the user's state space can be made smaller. Communication with the environment can be replaced by communication with the simpler abstraction presented by the agent.

3.3 The Informant

The Informant [7] is one of a growing number of information push technologies developed specifically for the World Wide Web. It manages standing queries for a user, running those queries against one of several Web search engines at fixed (user-specified) time intervals ranging from 3 days to several weeks. The results of a query are compared with previously run queries thereby identifying new web material if it exists. When new material is identified, the user is sent electronic mail (the push technique) informing the user that new material has been found. The newly discovered web documents are highlighted and ranked in a user-specific web page. Additionally, previously seen pages that have been modified since the last viewing are flagged for the user to see.

Again, this application involves a randomly changing environment (the documents on the web), a deterministic agent and deterministic user. This application differs from the smoke detector in many ways. The agent actually probes the environment checking for changes. The agent signals the user who then send a message to the agent for details. The user then engages in messaging with the environment (the web in this case), to check on those changes.

Figure 4 depicts the Informant agent operation in this environment.

3.4 Mail and News Filtering Agents

Mail and news filtering agents monitor incoming mail and news feeds for a user and perform actions such as routing, deletion, user alerting and automatic responses according to programmed or learned criteria [11, 15]. Routing means putting a document into an appropriate category for viewing and presentation. Filtering agents include traditional newspapers, magazines and television newscasts.

Figure 5 depicts the operation of a filtering agent in our conceptual framework.

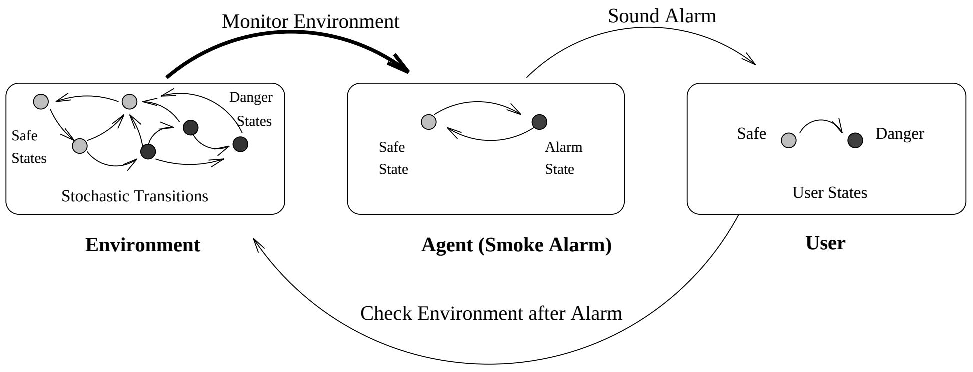

Figure 3: Smoke Alarm Schematic in Terms of the General Framework

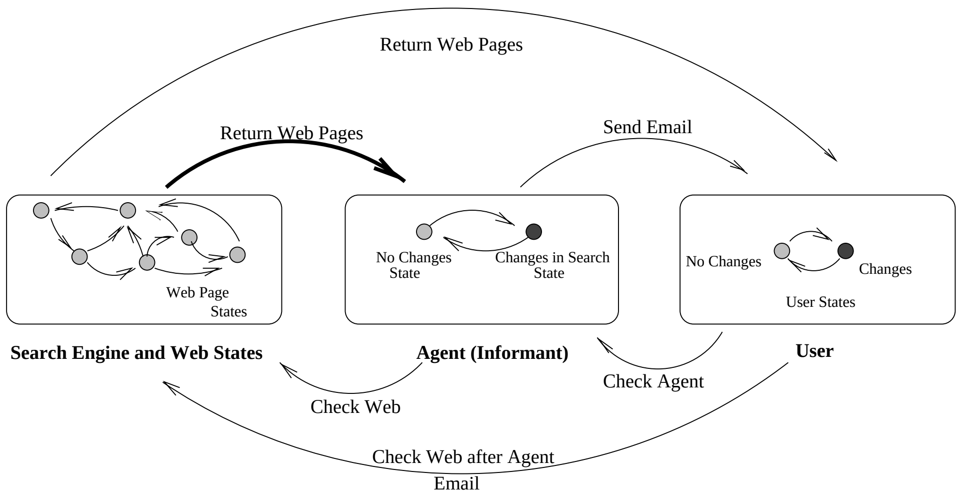

Figure 4: Schematic for the *Informant* Application

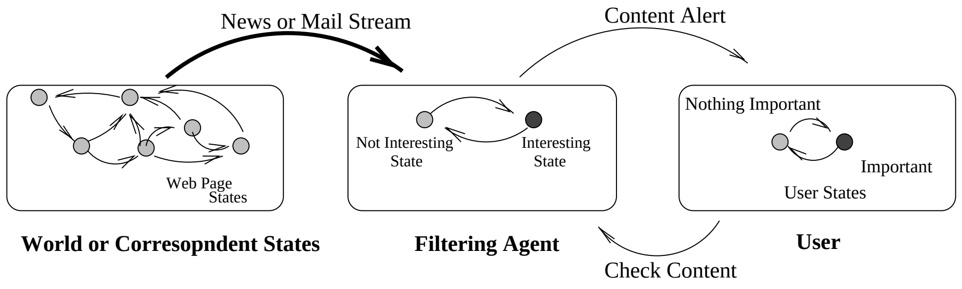

Figure 5: A Filtering Agent

4 Push and Pull

The concepts of push and pull are inherently related to the message traffic between user and agent, and the complexity of the messages sent by both parties. Further, the notion of state space complexity can be used as a measure of agent utility. Message traffic can be measured by the number of messages, their size (compressed or uncompressed), and their orientation (user to agent or vice-versa). State space complexity is more difficult to quantify properly but often very important to the notion of agent performance. For example, in the Informant application, the agent's state space keeps track of the time between updates whereas the user's state space does not. Succinctly put, agent technologies are most useful when presenting a simpler abstraction of the environment to the user. This is accomplished by reducing the number of states necessary for a user's model to perform adequately; moreover, information push technologies allow the user to redirect attention and resources which would otherwise be devoted to monitoring the environment.

We distinguish between two varieties of information push and pull. As is frequently the case in information science, we have no word to identify the attribute of a system which is "push-like" or "pull-like." Somewhat arbitrarily, we will refer to the attribute of "information flow" as being characterized *push* or *pull*. We describe "content information flow" "temporal information flow" in common-sense terms. The difference is straightforward; content flow roughly includes all information flow within single messages. Temporal information flow, on the other hand, is based on the proportion of total messages sent from user to agent and vice-versa. A situation in which there are many more messages sent from agent to user than from user to agent (such as with a smoke alarm or the Informant) is typical of a temporal push situation. Curiously, a technology will often be classified as "pull" in one domain and "push" in the other. A smoke alarm, for example, exhibits content pull (individual messages sent to the user are rather short) but temporal push (a single user request results in many agent responses). Conversely, consider a web search engine or a telephone "yellow pages" directory service. Both have small, simple requests that result in large preemptive replies, and therefore have large content push, but the symmetry in the number of user requests and agent replies is an example of temporal pull. We seek to define metrics which properly capture this behavior.

To this end, we introduce several notions of information flow that capture different aspects of push and pull technologies. Recall that $a_{\alpha\beta}(t)$ is a message sent from α to β at time t. This message content will act as a control on the recipient, β, and its transmission is triggered by the sender's state at that time. Therefore, $a_{\alpha\beta}(t)$ is a consequence of the state $s^{\alpha}(t)$ and will affect the state $s^{\beta}(t+1)$ in conjunction with $s^{\beta}(t)$.

Definition – The size of a message, $a_{\alpha\beta}(t)$, in bits is denoted by $|a_{\alpha\beta}(t)|$. The size of all messages transmitted from α to β between t_0 and t_1 is denoted by

$$s_{\alpha\beta}(t_0, t_1) = \sum_{t_0 \leq t \leq t_1} |a_{\alpha\beta}(t)|$$

and the number of messages over a time interval $[t_0, t_1]$ is denoted by $n_{\alpha\beta}(t_o, t_1)$. We will be using discrete time so that $t_1 - t_0$ is an integer.

Measuring message size using bits is problematic because of representational and compression issues. It is best to view message size quantification as application and model specific. For example, a smoke detector issues two types of messages – no audible signal or an audible signal. Accordingly, it might be appropriate to represent smoke detector messages as consisting of one bit per message although strictly speaking, the audible signal would require many bits to represent as an audio signal. Similarly, the decision to buy a newspaper can be represented using one bit per time interval. On the other hand, the number of bits communicated by a newspaper to a user clearly depends on the representation. Precise quantification of the size is best left to the modeler as appropriate for the setting.

With these notions, we can now talk about the difference between information *push* and *pull* technologies with some degree of quantification. In our notation, recall that $n_{UA}(t_0, t_1) + n_{AU}(t_0, t_1)$ is the total number of messages exchanged between the user and the agent in the time between t_0 and t_1. The following limit, if it exists, is the *push* index of an agent technology

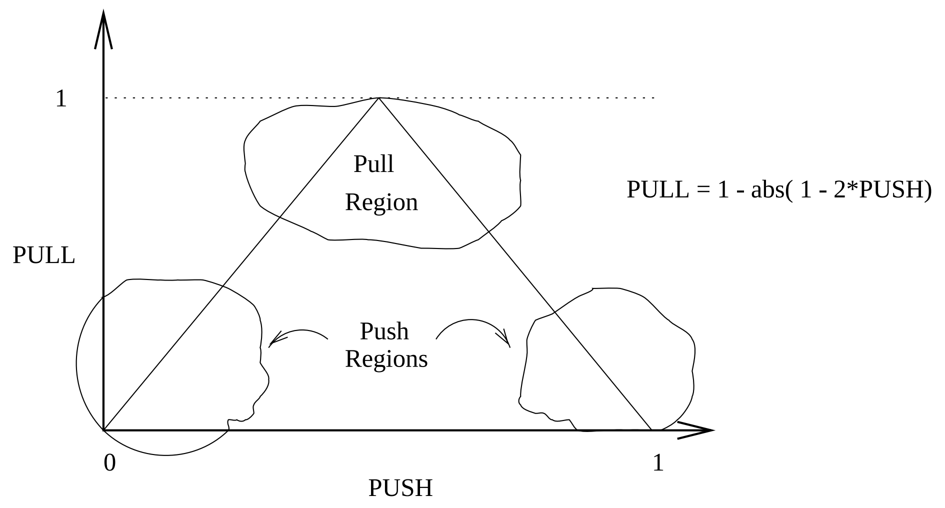

Figure 6: Push versus Pull Indices

DEFINITION – The Push Index, with respect to message count n, between α and β is

$$PUSH_n(\alpha,\beta) = \lim_{\substack{t_0 \to -\infty \\ t_1 \to \infty}} \frac{n_{\alpha\beta}(t_0,t_1)}{n_{\alpha\beta}(t_0,t_1) + n_{\beta\alpha}(t_0,t_1)}.$$

Clearly, $0 \leq PUSH_n(\alpha,\beta) \leq 1$. Moreover, we could replace the number of messages by the total sizes of the messages over time, leading to another concept $PUSH_s$. This would lead to another useful metric, $PUSH_s$. The push concept has a symmetry, namely

$$PUSH(\alpha,\beta) = 1 - PUSH(\beta,\alpha)$$

DEFINITION – An agent technology is a *push* technology for β with respect to the measure $n_{\alpha\beta}$ if $PUSH_n(\alpha,\beta) \approx 1$. It is a *push* technology for α if $PUSH_n(\alpha,\beta) = 1 - PUSH_n(\beta,\alpha) \approx 0$, which is equivalent to $PUSH(\beta,\alpha) \approx 1$.

It is important to emphasize that a push technology from one entity's perspective does not translate into a pull technology from the other entity's perspective. Temporal information pull is characterized by a symmetry between the number of user requests and the number of agent responses. For example, consider two email users. If user A is repeatedly sending unacknowledged and unsolicited mail to user B, this is a push situation. User A is pushing to user B and user B is experiencing information push not pull. If however user A asks for information from user B who acknowledges the requests and replies with answers, this is a pull situation for A. In this case, B is the source of the pulled information.

DEFINITION – The *pull* index of an agent technology with respect to the measure $n_{\alpha\beta}$ is defined by

$$PULL_n(\alpha,\beta) = 1 - |1 - 2 * PUSH_n(\alpha\beta)|.$$

We say that a technology is a pull technology if $PULL \approx 1$ and a push technology if $PULL \approx 0$.

Note that since $PUSH_n(\alpha,\beta) = 1 - PUSH_n(\beta,\alpha)$, we have

$$PULL_n(\alpha,\beta) \quad = \quad 1 - |1 - 2 * PUSH_n(\alpha,\beta)|$$

$$\begin{aligned}
&= 1 - |1 - 2 * (1 - PUSH_n(\beta, \alpha)| \\
&= 1 - |-1 + 2 * PUSH_n(\beta, \alpha)| \\
&= PULL_n(\beta, \alpha)
\end{aligned}$$

so that $PULL$ is symmetric in its arguments.

This notion of $PULL$ is just another way of saying that pull technologies correspond to situations in which the push index is approximately 1/2. Our measure for information push clearly identifies the recipient of the information push – if $PUSH_n(\alpha, \beta) \approx 1$ then β is being pushed and if $PUSH_n(\alpha, \beta) \approx 0$ then α is being pushed. On the other hand, if $PUSH_n(\alpha, \beta) \approx 1/2$ which is equivalent to $PULL_n(\alpha, \beta) \approx 1$ then it is not clear whether α or β is initiating the information requests. We will discuss this shortly in more detail.

Similar definitions exist for $PUSH$ and $PULL$ with respect to the measure $s_{\alpha\beta}$. Note that $PUSH_n$ and $PULL_n$ are defined in terms of the number of messages exchanged with no sensitivity to the sizes of the messages. By contrast, $PUSH_s$ and $PULL_s$ take into account the sizes of messages but not the overall numbers. Which is more appropriate depends on type of information flow being investigated. Typically, $PUSH_s$ captures content information flow, while $PUSH_n$ is better for temporal information flow.

EXAMPLE – Let us consider the newspaper example again. If a user issues an explicit request to buy a newspaper everytime then the measure $PUSH_n(\text{reader}, \text{newspaper}) = 1/2$ and $PULL_n(\text{reader}, \text{newspaper}) = 1$ suggesting that this is a pull technology. Since a newspaper contains much more than the 1 bit required to indicate a decision to buy the newspaper, we would have

$$PUSH_s(\text{newspaper}, \text{reader}) \approx 1$$

and

$$PULL_s(\text{newspaper}, \text{reader}) \approx 0$$

indicating that this is a push technology. So $PUSH_s$ captures our intuitive sense of this being a content push technology, and $PUSH_n$ identifies this as a temporal pull technology. If we were to subscribe to the paper, rather than issue an explicit request for each issue, we would have $PUSH_n(\text{reader}, \text{newspaper}) \approx 1$, reflecting the temporal push nature of the subscription. Web search engines are similar in terms of this quantification. The user requests and agent responses are equal in number but user requests use much fewer bits to describe than do the search engine's responses. Again, $PUSH_n$ captures the temporal information pull, while $PUSH_s$ captures the content push.

Given an agent capable of monitoring an environment on a user's behalf, the transferal of environmental information to the user can be implemented as either information push or information pull. Which method is more appropriate depends primarily on the accuracy with which the agent can anticipate the user's needs. If no model of these needs exists, the user must explicitly request the desired information. On the other hand, if user needs and the environment are well understood, the required information can be "pushed" to the passive user. Push agents can judge the value of the information to which they have access, and then choose whether to provide that information to the user or not. Therefore, when the user's needs can be accurately predicted, and the agent also has an adequate model of the environment, an information push agent is the logical choice, while filling more elusive user needs or monitoring complex environments is best implemented by information pull agents.

The results of choosing the wrong method are not difficult to find: a situation in which a push agent is used despite a poor model of user needs results in a number of "false alarms" and information overload. This is the generally the case in junk mail: an extremely imprecise model of user needs results in a great deal of wasted content. Conversely, when pull-type agents are chosen for tasks in which both user needs and the monitored environment are well understood, enormous time sinks result: users are required to request information that agents should have originally provided. Examples of this are more easily remedied than the first type. The desired information to be pushed is simply incorporated into the message. Marketers often use this to their advantage, as when the CD or book club requires you to respond to them if you do not wish to receive the "selection of the month." Here, user needs are well understood (namely, users generally do not want the "selection"), but the user is nonetheless required to participate in a useless exchange of messages. As another example, a newspaper subscription, as discussed above, provides a remedy for the problem of being required to issue a request for every newspaper. Generally, situations in which pull is used inappropriately involve agents which are not entirely operating in the user's best interests. These agents will eventually be displaced by virtue of the superiority of competing push technologies.

An important aspect of information push is that an agent can have an individualized model of the user, and can deliver information in accordance with the expressed needs of that model. This requires going deeper into the way information agent systems are implemented and evaluated. A simple quantifiable measure of implementation complexity can be obtained from the number of states that a user requires with and without an agent intermediary, while accomplishing the same degree of application performance. Note that this metric requires the user to be able to perform the task in question without the agent. These application performance measures will be described below.

In the meantime, let us simply assume that a user requires N_A states to operate in an environment with an agent intermediary, A, and requires N_U states without any specific agent technology. The performance of the user with respect to the environment, explored more completely below, is assumed to be the same in each case. The number of states required by the user in each case is a rough measure of the attentiveness needed to operate in the environment.

> Definition – Let N_A and N_U be the number of states required by a user with and without an agent technology to perform at an equal level in a given environment. Define the ratio
>
> $$COMPLEXITY_{AU} = \frac{N_U}{N_A}.$$
>
> This is the degree to which a user's equivalently performing state space can be reduced by using the agent technology. In general, N_U will be the same as the number of states required by the agent when computing N_A because the user will have to duplicate the agent functionality in order to achieve the same performance in the applications.
>
> Thus, $COMPLEXITY_{AU}$ being large means that the required state space complexity of a user is greatly reduced by using the agent technology. If the complexity measure is close to 1, there is relatively small reduction in the complexity of the state space required by the user even when using the agent.

Successful agent push technologies have a small $COMPLEXITY_{AU}$ meaning that there is a significant reduction in state space size by using the agent. Agent technologies in which $COMPLEXITY_{AU} < 1$ would require the user to have a more complex state space to accomplish a task at the same level of performance; such applications therefore have no practical value. It should be noted that situations do exist in which the user's state space is made more complex; agents may perform tasks which could not otherwise be accomplished by the user. In this case, the number of states necessary for the user to complete the task cannot be defined, so we cannot apply $COMPLEXITY_{AU}$.

> Examples – A traditional alarm clock reduces the user state space required to monitor the time before and after the alarm time by an enormous amount - from roughly the total number of states required to store time to only two states. Thus $COMPLEXITY_{AU} \gg 1$ for an alarm clock agent.
>
> Similarly, news and mail filtering agents reduce the required state space of a user. Intuitively, we believe that most successful "agent" technologies have a large complexity reduction measure.

We conclude this section by examining how our various examples rate using the metrics we have developed. In particular, note how the metrics $PUSH_s$ and $PUSH_n$ align with our common sense notions of how the information flows within single messages and over time.

Example	$PUSH_s$	$PUSH_n$	$COMPLEXITY_{AU}$	Common Sense (content flow)	Common Sense (temporal flow)
Smoke Alarm	0.5	1	large	pull	push
Yellow Pages	1	0.5	large	push	pull
Personal Email	0.5	0.5	medium	pull	pull
White Pages	0.5	0.5	large	pull	pull
Television	1	1	large	push	push
Junk mail	1	1	small	push	push
Newspaper/stand	1	0.5	large	push	pull
Informant	1	1	medium	push	push
Search Engine	1	0.5	large	push	pull

We have developed three quantifiable measures: $PUSH_s$, $PUSH_n$ and $COMPLEXITY$ and every agent can be described by the triple of numbers, called the *push/pull metric*:

$$(PUSH_s, PUSH_n, COMPLEXITY_{AU})$$

where $PUSH_s$ is the ratio of messages, measured in byte counts, that go between a user and an agent; $PUSH_n$ is the ratio between the number of messages, irrespective of size, that go between a user and an agent; and $COMPLEXITY_{AU}$ is a measure of complexity reduction accomplished by an agent from a users point of view, all as defined above. Our examples show that different push technologies can have quite different push/pull metrics.

5 Agent Performance Metrics

Push and pull describe the relative information flow between a user and an agent. Another fundamental concept, which we briefly alluded to above in the definition of $COMPLEXITY$, deals with the underlying performance of the agent technology in conveying the important information relative to the environment. Ideally, the agent will communicate the information most relevant to the user in dealing with the environment.

We will let $PERF(U, A, E)$ denote a general quantification of the user's performance measure of the agent system. This is distinct from the system's operating costs which depends on the number of bits transmitted, the number of states in the agent and user, and the frequency of communication.

$PERF(U, A, E)$ captures the overall agent system performance by averaging user costs over the environment state, $s_E(t)$, and the user state, $s_U(t)$. That is, there is a specific, state dependent cost associated with each pair consisting of user and environment states:

$$PERF_A(s_U, s_E).$$

These are averaged either over time or over the state spaces to obtain an overall application performance measure:

$$PERF(U, A, E) = \sum_{s_U, s_E} Prob(s_U, s_E) PERF_A(s_U, s_E)$$

or

$$PERF(U, A, E) = \frac{1}{N} \sum_{1 \le t \le N} PERF_A(s_U(t), s_E(t)).$$

The state of the user is affected by the messages that the agent communicates to the user. It may in fact be equivalent to the messages received by the user. When no message is sent by the agent, it is taken to be equivalent to a null message. Therefore, the state of the user is a function of the agent technology being used.

EXAMPLE – Information retrieval uses the notions of *precision* and *recall* to evaluate performance of a retrieval system [16]. These measures of performance are defined as follows. Suppose a user issues a search request, $q(t)$, at time t to a search engine resulting in a set $A_{q(t)}$ of documents while the "correct" response (as determined by experts) is a set $B_{q(t)}$. Then

$$PRECISION(q(t)) = \frac{|A_{q(t)} \cap B_{q(t)}|}{|A_{q(t)}|}$$

and

$$RECALL(q(t) = \frac{|A_{q(t)} \cap B_{q(t)}|}{|B_{q(t)}|}.$$

These are independent measures of performance, with each being equal to 1 indicating optimal performance. On the other hand, one measure can be close to 1 and the other close to 0.

An agent that is a retrieval engine accepts queries from the user, $q(t)$, and retrieves a set $A_{q(t)}$ of documents which is evaluated as above against the correct response, $B_{q(t)}$. The state of the user at time t is taken to be $A_{q(t)}$ and the state of the environment is $B_{q(t)}$. Accordingly, we can take

$$PERF_A(s_U(t), s_E(t)) = PRECISION(q(t)),$$

$$PERF_A(s_U(t), s_E(t)) = RECALL(q(t))$$

or some combination of the two.

The overall performance of the retrieval system is then, for example, captured by

$$PERF(U, A, E) = \frac{1}{N} \sum_{1 \le t \le N} PERF_A(s_U(t), s_E(t)) = \frac{1}{N} \sum_{1 \le t \le N} PRECISION(q(t))$$

where $q(t)$ is a sequence of queries. Of course, some combination of precision and recall are used together in real system evaluations.

The above measures of performance are closely related to the notion of Type I and Type II errors in classical hypothesis testing and decision theory [3]. Classical hypothesis testing develops tests for accepting or rejecting a hypothesis H_0 based on statistical evidence while decision theory derives a loss value for making incorrect decisions.

Let A^c denote the set complement of a set A. Note that

$$\frac{|A_{q(t)} \cap B^c_{q(t)}|}{|A_{q(t)}|}$$

is the probability that a randomly drawn document from $A_{q(t)}$ is not relevant. This represents a *false alarm probability.* Then

$$PRECISION(q(t)) = \frac{|A_{q(t)} \cap B_{q(t)}|}{|A_{q(t)}|} = 1 - \frac{|A_{q(t)} \cap B^c_{q(t)}|}{|A_{q(t)}|}$$

is the probability that a randomly drawn document from $A_{q(t)}$ is relevant – equivalent to the probability of not having a false alarm.

Similarly, noting that $A_{q(t)}$ is the returned set of documents (events), then

$$1 - RECALL(q(t)) = 1 - \frac{|A_{q(t)} \cap B_{q(t)}|}{|B_{q(t)}|} = \frac{|A^c_{q(t)} \cap B_{q(t)}|}{|B_{q(t)}|}$$

is the probability that a randomly drawn document (event) from $B_{q(t)}$ is not part of the returned document set, conditioned on the correct response $B_{q(t)}$. This can be viewed as the probability of a missed detection, conditioned on the correct response.

EXAMPLE – Figure 2 shows the structure of a smoke alarm system in our general agent framework. The environment's state space, $\mathcal{S}_E$, consists of a large number of states whose transitions are stochastic and extremely difficult to model. Conceptually, it is useful to think of the environment's states as being grouped into four sets: $\{s_{smoky-fire}, s_{false-alarm}, s_{missed-detection}, s_{safe}\}$. Transitions between these four states are probabilistic. When the environment is in states $s_{smoky-fire}$ and $s_{false-alarm}$, a message (eg. smoke) is sent to the detector. In the other two state groups, no message is sent.

The smoke detector (agent) has two states: $\mathcal{S}_A = \{s_{alarm}, s_{silent}\}$. The alarm is normally in the silent state, s_{silent} but upon receiving a "smoke" message from the environment, it transitions to the state s_{alarm} where it stays until the smoke messages cease. In the alarm state, the detector sends messages to the user. Otherwise, it does not.

Finally, the user state space has two states as well: $\mathcal{S}_U = \{s_{safe}, s_{danger}\}$. The user does not send messages to the detector or the environment in this simple model although we could of course make it more realistic by allowing the user to check the detector and environment directly. Similarly, the environment will not send messages to the user directly, using only the agent/detector as an intermediary. The detector system is reliable in the sense that it will transmit the appropriate messages to the user at all times and that the user will receive those messages unambiguously.

The reader should note the two distinct performance metrics that arise naturally in this system. The first is performance with respect to detecting danger (that is, fire) reliably from the user's point of view. This metric captures the success of the agent technology in solving the specific problem. The second is the amount of message traffic between user and agent, which indicates whether this is an information push or pull technology.

Both the user and agent are modeled by deterministic systems because they are essentially implementable as computer programs. Moreover, there is no benefit to implementing either of them stochastically. That is, we can ask if there is any improved performance to be gained by the agent (detector) randomly sounding an alarm or by the user randomly checking the environment. This is discussed below.

A smoke detector is meant to alert a user of fire. There are two types of errors again – a false alarm and a missed detection. Recall that the user is in one of two states depending on whether the smoke alarm is emitting a sound – the user either believes there is no fire (say $s_U(t) = s_{safe}$) or there is a fire ($s_U(t) = s_{danger}$). The environment consists of the four lumped states defined above – $\{s_{smoky-fire}, s_{false-alarm}, s_{missed-detection}, s_{safe}\}$ – two of which correspond to real fire and two of which do not.

A reasonable notion of $PERF_A(s_U(t), s_E(t))$ is then

$$\begin{aligned} PERF_A(s_U(t), s_E(t)) &= -C_1 \text{ if } s_U(t) = s_{safe} \text{ and } s_E(t) = s_{false-alarm} \\ &= -C_2 \text{ if } s_U(t) = s_{danger} \text{ and } s_E(t) = s_{missed-detection} \\ &= 0 \text{ otherwise.} \end{aligned}$$

Here C_1 and C_2 are positive loss constants. The maximal performance of the system in this framework is 0 (we are treating performance as a cost with negative costs meaning debits to the user.)

Since we are using $s_{false-alarm}$ and $s_{missed-detection}$ to denote environment states in which the smoke detector agent system fails, we can simplify matters. The agent to user communication is reliable so that

$$Prob(s_U(t) = s_{safe}, s_E(t) = s_{missed-detection}) = Prob(s_{missed-detection})$$

and

$$Prob(s_U(t) = s_{danger}, s_E(t) = s_{false-alarm}) = Prob(s_{false-alarm}).$$

Thus the expected cost of operating the system is

$$PERF(U, A, E) = -C_1 Prob(s_{false-alarm}) - C_2 Prob(s_{missed-detection}).$$

6 Stochastic and Learning Agents

We have been treating the environment as a stochastic entity but agents and users as deterministic. A natural question to ask is whether making agents' state transitions stochastic as well can improve their performance according to the metric $PERF(U, A, E)$ we introduced above. The answer in our framework is an unqualified *no*. We formally state a very general theorem to this effect next.

THEOREM – With respect to the infinite horizon, discounted cost performance measure (where $0 < \gamma < 1$ and $\mathcal{E}$ is the expected value with respect to both environment and agent states),

$$PERF(U, A, E) = \mathcal{E}\{\sum_{t=0}^{\infty} \gamma^{-t} PERF_A(s_U(t), s_E(t))\}$$

the optimal agent will be deterministic.

PROOF – The user-agent-environment system can be viewed as a controlled Markov process with state space $\mathcal{S}_U \times \mathcal{S}_E$. Costs are Markovian according to our definitions. The agent technology is the available control which can affect state transitions on the user and environment, again in a Markovian way. In this setting, a stochastic control policy can do no better than the optimal deterministic policy (see for example [9], Section 5.6).

This result must be interpreted carefully. The optimal policy or agent in such a framework is based on knowledge of the transition probabilities of the stochastic environment. With this omniscient view, the optimal agent will be deterministic.

However, in most situations a model of the environment and its statistics are not known. Computing an optimal policy or agent in such a case cannot be based on global knowledge of the environment. What is required instead is some form of sampling of both the environment and the user performance criteria. It can be shown that control systems with properly randomized sampling can adapt to achieve optimal performance which must in the end, by the above result, be deterministic in a stationary Markov environment. See [6] for example.

What this means is that agents operating in an unknown environment can use stochastic behavior to improve their performance but only if the stochastic behavior is used to adapt the control policy. In the end, the result of learning will be a deterministic agent state transition policy. Therefore, the above result is not inconsistent with the goal of developing learning agents in which some stochastic behavior would be appropriate.

Control and learning in nonstationary stochastic systems is very poorly understood, especially learning algorithms and convergence rates.

7 Summary

In this paper, we have introduced some quantification of agent systems and their performance. We have formalized the notions of information push and pull and have introduced a measure of agent performance with respect the user operating in a stochastic environment. Finally, we have discussed the value of agents behaving randomly, arguing that random behavior is not optimal in an absolute sense but might be necessary to *adapt* an agent to perform better in an unknown but stationary environment.

References

[1] A. Bensoussan. *Stochastic Control of Partially Observable Systems.* Cambridge University Press, Cambridge, UK, 1992.

[2] D.P. Bertsekas. *Dynamic Programming and Optimal Control.* Athena Scientific, Belmont, MA, 1996.

[3] D.H. Blackwell and D.A. Blackwell. *Theory of Games and Statistical Decisions.* Dover, New York, 1980.

[4] J.M. Bradshaw. *Software Agents.* MIT Press, Cambridge, MA, 1997.

[5] D.N. Chorafas. *Agent Technology Handbook.* McGraw Hill, New York, 1997.

[6] G. Cybenko et al. *Q-Learning: A tutorial and extensions.* Mathematics in Artificial Neural Networks and Applications, Oxford, UK, 1995.

[7] Informant. http://informant.dartmouth.edu.

[8] D. Kotz et al. Agent Tcl: Targeting the needs of mobile computers. *IEEE Internet Computing*, 1:58–68, August 1997.

[9] H. Kushner. *Introduction to Stochastic Control.* Holt, Rinehart and Winston, New York, 1997.

[10] H.R. Lewis and C.H. Papadimitriou. *Elements of the Theory of Computation.* Prentice-Hall, Englewood Cloffs, NJ, 1981.

[11] P. Maes. Agents that reduce work and information overload. *Communications of the ACM*, 37(7):30–40, 1994.

[12] R. Moore and L.L. Lesnick. *Creating Cool Intelligent Agents for the Net.* IDG Books Worldwide, San Jose, CA, 1996.

[13] A. Nerode. Linear automaton transformations. *Proc. Amer. Math. Soc.*, 9:541–544, 1958.

[14] D. Riecken. Special issue on intelligent agents. *Communications of the ACM*, 37(7):18–22, 1994.

[15] L. Spector. Automatic generation of intelligent agent programs. *IEEE Intelligent Systems*, 12(1):3–4, 1997.

[16] I.H. Witten, A. Moffat, and T.C. Bell. *Managing Gigabytes.* Van Nostrand Reinhold, New York, 1994.