A brief caveat before I begin: this paper is probably a bit more abstract and bewildering than the earlier papers I’ve composed for this seminar. This is not without purpose though: in borrowing from philosophy and moving onto unfamiliar ground, I find that I’m hard-pressed for lucid explanations. I’d like to ask that you bear with me, I think that this paper offers up some interesting concepts.

That said, I’d like to start this paper with a quick bit of history about one of my favorite philosophers, and his ideas. Nagarjuna (c. 200 AD) is a prominent figure in the history of Buddhist philosophy. He is credited with founding the “Middle Path” schools of Mahayana Buddhism and his consummate work, the *Mulmadhyamikakarika* (or *Fundamental Wisdom of the Middle Way*) is considered to be one of the most influential and also cryptic texts in all of Buddhist literature. The central topic of the *Mulmadhyamikakarika* is the Buddhist notion of “emptiness.” To western thinkers, emptiness is a somewhat difficult concept to grasp: it lies somewhere between the better-understood mainstays of realism and nihilism. Emptiness is the denial of independent, inherent, or essential existence of things. In truth, it is hard to make concrete statements about what emptiness actually entails for Nagarjuna urges his readers to consider emptiness itself empty.

My own interpretation of emptiness deviates slightly from the traditional Buddhist conception. I view empty things as items with no “definite” state, entities that could be considered to be:

- In a continual state of flux
- In a constant process of adaptation
- Without essential properties

---

What I’d like to consider in this paper is the software analog of “emptiness” and what its ties to constructive and extensible media. I assert that if an empty model is adopted for software, then extensible or socially constructed interfaces follow of necessity. This would be an interesting shift from the current state of affairs where most application interfaces are typically of the one-size-fits-all variety and don’t change more than once every few years.

Let’s continue by considering what properties non-extensible media and interface generally exhibit. Most modern applications (multimedia or otherwise) are the antipode of Nagarjuna’s conception of emptiness. They are intentionally designed to be generic and for generic users: “Much of interface design is built around the premise of ‘the typical user.’” This means that modern non-constructivist applications have inherent and essential natures. That is, these systems are developed so that there is a singular application, (Word for instance) which we can accurately specify details about. These applications are fundamentally the same from one computer to the next. They allow a small degree of personalization, but each installation of Word has the same degree of personalization, and the same essential properties.

As an aside, I assert that the myth of the typical user has led to poor, depersonalized, one-size-fits-all interfaces. The conclusions of a study conducted to assess the implications of designing for culturally diverse users support this assertion:

While issues in cultural and gender diversity are vitally important to the future of human-computer-interaction, they cannot be addressed by prejudicing the results with overly-generalized characterizations of user populations and vague, unsupported guesswork about what specific user populations find to be an “appealing perceptual experience.”

---


Our systems and interfaces depend upon “overly-generalized characterizations” of our users. An interface designer forms a model of the typical user, and then constructs an interface for this abstraction. This approach tacitly forces conformity of the user to designer’s vision of a typical user. If users drastically deviate form our preconceived notions of what their role should be within the workspace, then they are precluded from effectively using our software.

So now let’s turn our attention to empty systems. An empty system has no essential nature. As I noted earlier, this means that the system is in a constant state of flux and adaptation, and one instance of an empty system would ideally be different from all other empty systems. This would mean that my empty version of Word would be completely different from any other person’s and would also be different from one moment to the next. Borrowing from the ancient Greek philosopher Heraclitus the Obscure, we could state that you “never use the same application twice.” Admittedly this could be disorienting for users of traditional applications as these systems are not only the same from one moment to the next, but are also quite similar to other applications. Interface guidelines like those published by Microsoft and Apple help ensure this sort of homogeneity and standardization.

How could empty systems be created? The first necessary component would be some sort of abstraction for the individual user’s environment, a reflection of the individual’s specific empty environment. In computer jargon, this typically termed a user profile. If the interface is the only part of the application that is empty, then you need a separable interface system that makes a clean delineation between the functional engines of the software and the adaptable and changeable interface portions. A more detailed discussion of these technologies and their implementations is available in “A Critical Examination of Separable User Interface Management Systems: Constructs for Individualization.” User profiles and separable user interface systems are becoming more and more common, in fact “n-tier application architectures” (which are all the

rage in development circles right now) are essentially separable interfaces. User profiling systems are also becoming increasingly significant. Microsoft’s recent acquisition of Firefly Networks was undoubtedly motivated (in part) by Firefly’s profiling and personalization technologies.

These constitute only one portion of our hypothetically “empty” system though. We need some sort of source for both flux and adaptation. The biological metaphor for information that I introduced in “Information Metaphors” could provide the necessary source of continuous variability. Basically, the computer interface or the multimedia content would be continually adapted and changed in accordance to the user’s ecosystem (or profile), which would itself be continually updated as new information about the user is collected.

One missing component still remains, however: if our system is continuously introducing new content or interface components, where would these come from? This is where the socially constructed aspect of empty systems comes into play. Ideally, content and interface components would come from the largest imaginable source (gene pool): the Internet and the many people who produce its content. An empty system would take content or components that are outdated or have been discarded (because of criteria stored in the user’s profile) and exchange them for new content or components.

So, why on earth would anyone want a system that is constantly changing and isn’t really the same from one moment to the next? At least for some content systems, like news and information systems this makes good sense. An empty news system would be, to my mind, ideal; it would continuously stream in new information that is constantly adapted to my current interests. When my interest in a topic begins to wane, it becomes extinct in my empty ecosystem and is replaced by new species of information that will compete for my attention.

Let’s look at systems where the use of extreme adaptability and fluxism is a little less inviting. A computer operating system interface could certainly be made to be empty. Would this be
something that is desirable though? I think that, thought about in a certain light, it is. Consider that right now we force users to learn and adapt to all the different programs that they come across. If an empty system is implemented effectively, I think an interesting inversion would occur: the computer would do the adapting instead of the user. Consider what happens when a user first upgrades to a new version of an operating system that contains drastic interface changes. If an individual were using current non-empty systems they would be overwhelmed with a dizzying array of unfamiliar features and commands. An empty ecosystem could gradually, little-by-little, ease a user into a more comfortable relationship with the new components that are available. An empty system could also provide a highly-individualized experience. The generic aspects of current interfaces that I critiqued earlier could certainly be remedied by an empty environment.

A system does not have to be fully empty though. Perhaps some sort of synthesis between current system and empty systems would be more desirable. I have found that, at least, the conception of an empty system is an interesting alternative to current system design methods.