Intelligent Context Agents: A System for Smart Recommendations

Daniel Afergan, Gabriel Schine, Ross Wang, Alexandre Zani,
Marco Zamarato, David Murphy, Michael Gilbert, Jeffrey Nichols
{afergan, thatguy, rosswang, azani, zamarato, djmurphy, mdgilbert, jwnichols}@google.com
1600 Amphitheatre Parkway
Mountain View, California 94043

Abstract

Intelligent digital personal assistants have become ubiquitous, as now all major smartphones and smartwatches have voice-based systems that can perform actions for the users and home voice systems, such as Amazon Alexa and Google Home, offer ever-present and attentive assistants. However, these devices know very little about what the user is doing and can only perform explicit actions that the user specifies. Intelligent agents that can combine sources of user information and a model of the user can remove this limitation by allowing a system to know more about a user and the user's current state and react accordingly. We discuss the possibility of a system where context agents can combine signals and create user models in order to develop smart and automatic recommendations, and question how an intelligent suggestion system can best be developed and integrated. We propose several questions about how this system could operate.

Introduction

Smart assistants seek to automate or help a user perform actions on a regular basis, or provide suggestions as to what the system thinks the user wants to do. However, the productivity of these assistants is currently limited. The bottleneck comes from only being able to respond, retrieve, and react to the user's input or defined entities (such as calendar entries). A typical user with a smart device interacts with a number of environmental variables, and would expect that the system reacts differently to his or her state. However, the user cannot explicitly tell the system all of these factors, and any effort spent helping the system understand the user's current state limits the benefits of the system.

In order for a smart assistant to truly understand the user, they must be cognizant of the user's context. Dey defines context as "any information that can be used to characterize the situation of an entity ... that is considered relevant to the interaction between a user and an application, including the user and applications themselves" (Dey 2001). Since a user would be unable to explicitly inform the system of all of these changes, the system must be able to infer it, and use this knowledge to help guide actions and suggestions. The system must maintain a model of the user, and then use data to best predict what the user might want to do.

One generalized assistant would either be too generic or too complex to service users who have different applications installed, devices, or use cases for their systems. Instead, we propose that an intelligent recommendation system could be broken down into *context agents*, services with access to user data that can yield additional data or create propositions for user suggestions or system actions. These agents could incorporate machine learned models of the user (or an aggregate model of all users) in order to create suggestions, or a service that uses the model to decide which suggestions would be most useful.

As smart assistants become commonplace, with all major smartphone operating systems and even some platforms having their own assistants (Siri, Google Assistant, Cortana, Viv, M), as well as voice-receptive smartwatches and home voice assistants such as Amazon Echo and Google Home becoming more popular, it is imperative that context agents aim to understand the user and what the user is doing. While the current systems can react to direct commands and perform a limited set of actions, by combining context with machine learned models of the user, we can improve suggestions and system experience overall.

We believe that in order for these intelligent agents to improve user experience in a system, they must be extensible and still offer a natural conversational user experience. Most popular modern operating systems derive a lot of their information from applications that were built by someone other than the system manufacturer. Because of this, an intelligent suggestion service must allow these third party elements to access the system, and give users access to these . Extensibility also introduces development questions, such how it should be structured, what the properties of those agents might be, and how we can make sure they behave relatively well.

We propose six questions that developers, designers, and user experience professionals must consider when creating this framework and implementing these assistants.

- What types of agents exist?
- Where do agents run?
- How do agents train and personalize?

- How do we ensure that agents do not hurt user experience overall?
- How do we standardize data types and concepts?
- What happens when an experience fails?

Related Work

Context-aware computing or software refers to systems that can adapt to the execution environment or factors such as location, nearby people, hosts, accessible devices, physiological and affective states, personal history, daily behavioral patterns, and changes to these over time (Satyanarayanan 2002; Schilit, Adams, and Want 1994). Although this concept is not novel, recent changes in multi-device and ubiquitous computing, improved device sensors and power, the use of applications which all have user data, and cloud storage and computing have all led to the possibility of a system which can aggregate user data from many sources in order to create a user model and use this model to power recommendations.

Recent efforts have looked about what information from the device or user state can predict user state or interruptibility, including device or application usage and interruptibility. These variables are particularly useful because an intelligent assistant must be able to anticipate what the user will want to do, and when the user can be interrupted.

Welke et al.'s analysis of app usage differentiated showed that almost all users had unique patterns of usage, but Zhao et al. inspected mobile application usage over time and discovered a set of 382 distinct types of users (Welke et al. 2016; Zhao et al. 2016). Although these findings conflict, they signify that a system might be able to build a model of app usage for each user, but also leverage aggregate models to help prediction. Newman et al. accompany app usage with sensor data (Newman et al. 2010) to determine usage scenarios, and Beach et al. propose that incorporating social network data may allow the system to understand the user better and incorporate preference-aware suggestions (Beach et al. 2010).

In addition to understanding the user and which application the user may want to deploy, an effective system must also know when to make suggestions to the user. In order to this, a context agent must be able to predict user interruptibility. Hudson et al. used a machine learning model of the environment to predict this state, and Kern et al. used accelerometers as well as location and audio sensors to determine user activity and when to intervene (Hudson et al. 2003; Kern et al. 2004). In addition to environmental factors, information about user state can also add context awareness to a system. Systems have been designed to use physiological state to change notification settings, such as phone volume and messenger system status based on heart rate variability, pupil dilation, EEG signals (Bailey and Iqbal 2008; Chen and Vertegaal 2004; Iqbal, Zheng, and Bailey 2004). Researchers have also found success queuing messages when a user is busy and delivering them once the user is in a better contextual or cognitive state (Drugge et al. 2004; Horvitz 1999).

This contextual and interruptibility information can be used as continuous input to context agents and interactive recommendation systems, making the systems more in sync with the user, providing appropriate help and suggestions when needed. However, contextual data is different from direct input modalities because the user cannot directly control his or her own inputs, nor does the user know all of the inputs that are being fed to a model. To effectively use contextual data, the interactive system must be carefully designed to take advantage of this more subtle new class of input, leading to *implicit interfaces*. These are systems that rely on passive or implicit inputs, which are user actions or situational contexts that the system understands as input, but that were not actively chosen by the user to interact with the system (Schmidt 2000). Context agents must consider the properties of the sensor data as well as the user states that the data implies.

Agent Considerations

We consider a future system where context agents are rule or model-based and provide suggestions to a recommendation system. Although current mobile operating systems such as Android and iOS currently rely on explicit rules, newer operating systems could directly support context agents and intelligent recommendations. However, in order for these suggestions to augment user experience instead of harming it, many factors must be considered.

What Types of Agents Exist? There might be many types of context agents in this system. Some agents might provide context such as sensor data or user actions to the system, some might interpret these low-level signals into higher-level concepts, and some might focus just on proposing suggestions. Should these agents be distinguished and handled differently by the operating system? Can they cross boundaries and perform multiple functions? In addition, we must consider the granularity of agents, and whether there should be a few agents capable of delivering many types of proposals and suggestions, or whether there should be a large network of agents who each specialize in small tasks.

Where Do Agents Run? Ideally, agents can run on a local device so that they have access to all user data. However, this requires a lot of processing power, and this means that there must be synchronization for multi-device scenarios. Cloud processing could offload the processing, power, and memory requirements of these agents running constantly, but would require devices to be online and to constantly pass context data. In addition, it would either require a central source that can run agents, or for each agent's service to have cloud computing resources. How can the operating system optimize the usage requirements for agents?

How Do Agents Train and Personalize? Agents must balance creating generalizable suggestions with suggestions for the specific user. Does the burden of the user model fall upon each agent, or can a suggestion service instead decide which agents and suggestions are most useful for

that particular user? In order for agents to train models, they must track historical data– do they store this themselves, or does the system require a log of past context data and usage for training? Do agents get feedback and know when their suggestions are useful? The operating system must consider how to best track training data and how to treat training data versus current data.

How Do We Ensure That Agents Do Not Hurt User Experience Overall? Although the aim of agents is to improve user experience and increase the bandwidth from the user to the system, the operating system must must be careful to make sure that they add a positive value instead of straining user experience. Constant agent processing can monopolize memory, slow processing speeds for other processes, or hurt battery life, which are all major user concerns. Scheduling agents could help alleviate these concerns and provide flexibility. In addition, users care about their privacy and knowing what data is being shared with which services. With so many contextual variables being tracked, a system will need to consider how to protect a user's privacy and allow a user to control personal data, without being burdensome. In addition, the system must be able to provide timely suggestions that help the user, without constantly interrupting the user and becoming a nuisance.

How Do We Standardize Data Types and Concepts? In order for agents or the system to share data in a useful way, the entities must agree on a format and structure. While system concepts such as GPS location or current device state can be formalized, agents and applications will need to agree on how to express higher-level context such as where the user is, what they think the user is doing, or if the user is busy. In addition, multiple agents could provide conflicting data, which will need to be handled.

What Happens When an Experience Fails? In this seamless integration of system, agents, and applications, when something goes wrong, how does the user know where and why the failure occurred? It is important to provide transparency and system status to the user, especially when the user expects that an automated action or action that does not affect the UI has occurred. The system must be able to recognize the cause of failure in order to prevent future occurrences. Additionally, in order to preserve trust, it must limit the harm, allow the user to know the state of the system and recover, as well as provide feedback which can prevent future failures.

Discussion

As operating systems become more powerful and try to assist users with more granular tasks, these platforms should build models of users and integrate intelligent assistants and suggestions. Current assistants are static and do not consider a user's individual preferences nor their context. By integrating knowledge about the user and the user's state, these systems can become powerful and more useful. We believe that the future of assistants is to passively and safely monitor the user and to improve performance without effort on the user's behalf or harmful interruptions.

We highlight some of the technical and user experience issues that are important when creating an predictive system. These systems must access and integrate knowledge from different entities and applications of the system, and be able to then perform actions with these applications. Judicious use of suggestions and automations for the user will help the system provide an overall benefit that outweighs any costs incurred by the system.

Bio

Daniel Afergan is a software engineer at Google working on intelligent recommendation systems. His Ph.D. research focused on implicit input and adaptive user interfaces.

The other authors are software engineers, research scientists, designers, user experience engineers, and user experience researchers.

Our preferred presentation format is a presentation, but we could also present a poster.

References

Bailey, B. P., and Iqbal, S. T. 2008. Understanding changes in mental workload during execution of goal-directed tasks and its application for interruption management. *ACM Transactions on Computer-Human Interaction* 14(4):1–28.

Beach, A.; Gartrell, M.; Xing, X.; Han, R.; Lv, Q.; Mishra, S.; and Seada, K. 2010. Fusing Mobile, Sensor, and Social Data To Fully Enable Context-Aware Computing. *Workshop on Mobile Computing Systems & Applications* 60–65.

Chen, D., and Vertegaal, R. 2004. Using mental load for managing interruptions in physiologically attentive user interfaces. *Extended abstracts of the 2004 conference on Human factors and computing systems - CHI '04* 1513.

Dey, A. K. 2001. Understanding and using context. *Personal and Ubiquitous Computing* 5(1):4–7.

Drugge, M.; Nilsson, M.; Liljedahl, U.; Synnes, K. r.; and Parnes, P. 2004. Methods for interrupting a wearable computer user. In *Eighth International Symposium on Wearable Computers*, volume 1, 150–157. IEEE.

Horvitz, E. 1999. Principles of Mixed-Imitative User Interfaces. *Conference on Human Factors in Computing Systems* (May):159–166.

Hudson, S. E.; Fogarty, J.; Atkeson, C. G.; Avrahami, D.; Forlizzi, J.; Kiesler, S.; Lee, J. C.; and Yang, J. 2003. Predicting human interruptibility with sensors: a Wizard of Oz feasibility study. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, volume 12, 257–264. ACM.

Iqbal, S. T.; Zheng, X. S.; and Bailey, B. P. 2004. Task-evoked pupillary response to mental workload in human-computer interaction. *Extended abstracts of the 2004 conference on Human factors and computing systems CHI 04* 1477.

Kern, N.; Antifakos, S.; Schiele, B.; and Schwaninger, A. 2004. A model for human interruptability: experimental

evaluation and automatic estimation from wearable sensors. In *Eighth International Symposium on Wearable Computers*, volume 1, 158–165. IEEE.

Newman, M. W.; Ackerman, M. S.; Kim, J.; Prakash, A.; Hong, Z.; Mandel, J.; and Dong, T. 2010. Bringing the field into the lab: supporting capture and replay of contextual data for the design of context-aware applications. *UIST '10 Proceedings of the 23nd annual ACM symposium on User interface software and technology* 105–108.

Satyanarayanan, M. 2002. Challenges in implementing a Context-Aware System. *Pervasive Computing* 2.

Schilit, B. N.; Adams, N.; and Want, R. 1994. Context-aware computing applications. In *IEEE Workshop on Mobile Computing Systems and Applications*, 85–90.

Schmidt, A. 2000. Implicit human computer interaction through context. *Personal Technologies* 4(2-3):191–199.

Welke, P.; Andone, I.; Blaszkiewicz, K.; and Markowetz, A. 2016. Differentiating smartphone users by app usage. *UbiComp '16* 519–523.

Zhao, S.; Ramos, J.; Tao, J.; Jiang, Z.; Li, S.; Wu, Z.; Pan, G.; and Dey, A. K. 2016. Discovering different kinds of smartphone users through their application usage behaviors. In *Proceedings of the 2016 ACM International Joint Conference on Pervasive and Ubiquitous Computing - UbiComp '16*, 498–509.