BLOG@CACM

The *Communications* website, http://cacm.acm.org, features more than a dozen bloggers in the BLOG@CACM community. In each issue of *Communications*, we'll publish selected posts or excerpts.

twitter
Follow us on Twitter at http://twitter.com/blogCACM

DOI:10.1145/3479972 http://cacm.acm.org/blogs/blog-cacm

New Life for Cordless Communication, Old Regrets for Software Projects

Andrei Sukhov and Igor Sorokin ponder the potential benefits of DECT to the Internet of Things, while Doug Meil considers how software engineers should reflect on their accomplishments.


Andrei Sukhov and Igor Sorokin

A Chance for Rebirth

https://bit.ly/3lkg9lf
July 1, 2021

The Internet of Things (IoT) is a new area of infocommunication technologies including not only home appliances with an IP address and Internet control, but also a variety of industrial technologies. These technologies need to be developed at all layers of the OSI model, but the need for security mechanisms should be emphasized.

At the physical level, wireless technologies (Wi-Fi, Bluetooth) have significant range limitations. The few tens of meters these standards allow are not enough. Alternative wireless technologies LPWAN (Low-power Wide-area Network) and NB-IoT (Narrow-band Internet of Things) are rapidly developing and used in many areas. Their range is an order of magnitude higher, but with no generally accepted standard, they are regulated by private companies. Yet unresolved issues remain, so attempts are being made to develop new technologies to lay the foundation of the IoT.

Some older wireless technologies are well studied but in low demand, as interest in them has passed. We can try to give such technologies a second life in IoT, such as DECT (digital enhanced cordless telecommunication), which operates in the 1.9GHz band.

After the emergence of the GSM standard, interest in DECT dropped. An attempt to use DECT as a physical layer for TCP/IP was unsuccessful, due to low transmission rates. Emerging demand for IoT technologies gives DECT a chance for rebirth, as it can increase the communication range between devices up to several hundred meters and achieve a data transfer rate of several hundred kilobits per second, sufficient for transmitting control and monitoring information in real time.

Note the advantage of DECT in comparison with existing standards for the IoT's physical layer. The frequency band in which it operates lacks many different devices, compared to the 2.4GHz band and the ISM (Industry, Science, Medicine) bands. In 2020, the EU adopted the DECT-2020-NR standards package to support IoT applications[1]. Before that, in 2017, the DECT ULE (Ultra Low Energy) standard was adopted.

The advantages achieved when using DECT technology include:

1. Increased action at a distance of up to 600 meters.
2. Availability of ULE technology to extend service life.
3. Technologies to protect data transmission implemented at the physical layer of the OSI model (radio path level).
4. The ability to build a network infrastructure for mobile terminals without losing communication during transitions between base stations (handover).
5. A large number of standards approved by ETSI in 2020.

An IP over DECT data network should be a key component of a full-fledged family of IoT technologies. The ability to transfer data over the IP protocol, and to assign an IP address to a subscriber terminal, are the basic elements of the proposed concept. VoIP support has long been included in the DECT standard.

In some implementations of DECT phones, the subscriber device is an Android device that supports the TCP/IP protocol stack. The device is assigned an

IP address and can use all the capabilities of the IP protocol. There are many IoT MQTT (Message Queue Telemetry Transport[2]) implementations available for the Android platform, making testing easier. A DECT base station can be implemented based on a standard Linux server with a specialized PCI card.

In the cases described here, there is no need for high data transfer rates, and the speed at which voice transmission is organized is sufficient for IoT tasks.

The market presence of Android devices with DECT support makes it easy to assemble an experimental bench for tests and measurements. The authors anticipate the appearance of DECT subscriber devices based on minicomputers with a DECT module running Linux and Android operating systems. The emergence of such devices will mark the beginning of development as full-fledged technologies for IoT. The simplest smart plug devices based on DECT are already on the market, such as AVM's FRITZ! DECT technical product line.[3]

Our team plans to assemble a full-fledged DECT over IP stand, and to start developing and testing various IoT technologies. We are open to cooperating with other research groups on this.

References

1. ETSI. DECT-2020 New Radio (NR); Part 1: Overview; Release 1. European Telecommunications Standards Institute, Technical Specification (TS) 103 636-1, July 2020.
2. Boyd, B. et al. Building Real-time Mobile Solutions with MQTT and IBM MessageSight. *IBM Redbooks*, 2014.
3. https://en.avm.de/products/fritzdect/

Doug Meil
Software Learning: The Art Of Design Regret

https://bit.ly/2TNkzFN

August 2, 2021

"What If?" is a question so fundamental to human learning that it has infused generations of science fiction enthusiasts with the possibilities of fixing our mistakes through time travel; 19th-century writer H.G. Wells' *The Time Machine* and "Star Trek"'s City on the Edge of Forever episode come to mind. Given time machines are not on the horizon, the best we can do is to look backward for insight and apply lessons forward. This is not as easy as it sounds, as there are two equally unhelpful poles: the first is to never to look to the past and question what could have been improved, and the second is to persistently ruminate in the past. The goal is to live somewhere in the middle. How should software engineers try to classify their reflections?

Retrospectives have long been part of software engineering practice. It can be tempting to look at prior efforts and label every decision a "flaw" or "bug" if it doesn't agree with one's sensibilities. This is not constructive; understanding the context of decisions is critical, since no effort exists in a vacuum. Factors such as budget, resources (in quantity, quality, experience, and personality), technical options, and schedule all affect the decision-making process. Only by understanding design context can we differentiate between the preventable and the unavoidable, and understand what we could reasonably beat ourselves up about when we need to just let go.

Anachronistic Regret

This is where a framework or technology options did not exist at the time of a design decision, but regret is felt for not having those options anyway. While this can make for some interesting hypothetical discussions, such as the effects PCs could have had on the 1960s space race, it can also be taken too far, such as pondering how Abraham Lincoln could have revolutionized space travel as President if he only had rockets and computers during his administration. There was no decision possible, because there were no valid options at that time.

Actual Mistake Regret

It happens, where "it" can be everything from fat-fingers, code-horrors, or bear-traps, and Refactoring (Fowler). An example I lived through was a colleague enamored with the Inversion of Control and mocked objects pattern; there were unit tests, but of mocked objects instead of the codebase. When the codebase was deployed, it was a disaster, and I had to clean it up. The issue was not that Inversion of Control and mocked objects were not legitimate patterns; they were taken too far, as adherence to patterns became the goal, instead of the functionality of the overall software effort.

Decision Regret

This is the regret of the "road not taken." In software efforts, there are design choices that seem ever-pitted against each other, such as natural vs. synthetic keys for database design. Yet there are plenty of other cases where multiple reasonable options could exist for a situation that are not so doctrinally charged, each valid and "appropriate enough." As long as each option was evaluated honestly and thoroughly in terms of strengths and weaknesses—and, ideally, documented—this is really the best we can expect any software engineer to do. Decision regret is an inevitable outcome of making decisions, and it is better to make progress and live with Decision Regret than to be paralyzed by it.

Unknown Consequences Regret

This is the case of "I wish I would have known that at the time," where one experiences an unanticipated side effect or edge-case of a design, which often pops up at the worst possible time. These do not necessarily invalidate an overall design, but expose extra conditions that need to be addressed. The Java programming language is filled with these, particularly around memory management and garbage collection. Java has proven itself effective in a great many cases, but it also contains some surprises for designs that require operating under high memory load, where software solutions tend to work… until they don't. This necessitates diving into arcane Java Virtual Machine settings, and sometimes redesigning some software elements in response. In fairness to Java, every programming language and technology framework has sharp edges lurking somewhere, and finding those edges is the frequent consequence of doing interesting work.

Missed Opportunity Regret

Who hasn't exclaimed, "Why didn't I think of that?" Well, you didn't, and that's life. Again, the best one can do is to continually strive to expand one's knowledge horizons and look for opportunities to apply those lessons in the future. Fortune favors the brave—and the prepared. Iteration is equally important, as the more one practices, the better one can become at pattern recognition.

Andrei Sukhov is a professor and head of the Network Security Research and Study Group of HSE University, Moscow, Russia. **Igor Sorokin** is a postgraduate student in the Department of Computer Engineering of HSE University, Moscow, Russia. **Doug Meil** is a software architect at Ontada.