Author Notes

- ✔ The type has been replaced in your figures; please proof the figures carefully for any errors.
- ✔ The last page contains pull quotes that might be used in the article, depending on design and spacing considerations. Are these pull quotes appropriate? If not, please provide several alternatives. Pull quotes should be concise and taken verbatim from article text.

The Monkey, the Ant, and the Elephant: Addressing Safety in Smart Spaces

Sumi Helal, Lancaster University

Smart spaces deliver digital services to optimize space use and enhance user experience. The impact of ill-programmed applications in such spaces goes beyond loss of data or a computer crash; there is the potential risk of physical harm to the space and its users. Ensuring safety in this type of cyberphysical system is critically important.

Smart spaces[1] such as smart homes, comprised of Internet of Things (IoT) devices, people, and physical content, are different from traditional computer systems. Their cyberphysical nature ties intimately with the users and the built environment. Errors and conflicts in such spaces could have harmful, dangerous, or undesired effects on the user, the space, or the devices making up its IoT. Little has been done to date to ensure that smart space programming models or runtime systems are capable of handling such critical safety issues. In this column, I will shed some light on the important need of accelerated advancements in these technologies, given the highly anticipated proliferation of the IoT and especially the personal IoT. When deployed within users' homes, the personal IoT will constitute the most popular and perhaps the largest category of future smart spaces.

SMART SPACES

<AU: Kindly check that the section heading is appropriate here.>

I want to start by visiting the remote garage door opener popular in North American homes. It represents an example of a simple smart space (smart garage) that consists of a powerful actuator (the 0.25–0.5-hp motor drive), a remote portable open/close switch, a light, a timer to turn off the light, and two infrared motion detection sensors installed a few inches off the ground at both ends of the garage door.

Digital Object Identifier 10.1109/MC.2020.2978378
Date of current version: xxxxxx

0018-9162/20©2020IEEE

EDITOR **DIMITRIOS SERPANOS**
ISI/ATHENA and University of Patras; serpanos@computer.org

This simple smart space is successfully used by tens of millions of people every day, thanks to its reliability and safety features. It can be operated from a distance, automatically turns the garage light on, and, most importantly, reliably detects the smallest objects or people, like small children, who may be in the door's path while closing. during door closure, which gets aborted through a reversing mechanism. **<AU: Please check whether the preceding edited sentence conveys the intended meaning.>** These safety features were possible under a fixed, nonprogrammable design of all of the involved devices and sensors making up the garage door opener. Nothing could interfere or alter the operational logic of the garage door opener system, which guarantees its safety. But there is much more than a garage door opener in a smart space, and without the ability to program, reprogram, or download new applications, the space would be of very limited smartness and utility. For this reason, several efforts began almost two decades ago to find successful models for "programming pervasive spaces."[2]

One model that has been very successful is the service-oriented device architectures (SODA) model,[3] which enabled the programming of smart spaces as service composition over software services representing all elements of the space. This ranged from full-fledged computers to mobile devices, actuators, or even pinhead-sized small sensors. Replacing the rigid and expensive ad hoc system integration approach to creating smart spaces by SODA's flexible, (re)programmable approach was disruptive in two key aspects. First, it created a much-needed separation between a pervasive system and its pervasive applications. Second, by creating an external representation of every element in the space in the form of an active service on the space edge computer or the cloud, it was possible to engage software developers in creating such applications instead of relying on system integrators and engineers. Several implementations of SODA are in place today, including the Atlas sensor platform and middleware.[4]

Soon after celebrating the successful SODA milestone, it became evident we just created an unrestrained programming model that lacks adequate safety guards. That is, we realized that SODA was too powerful to be safe. We also realized that while its conceptual framework is elegant, SODA did overpromise a little.

THE MONKEY

First comes the monkey. By abstracting devices, sensors, and actuators into services, not only do we manage to engage software developers, but we overpower them to do just about anything with these services through unrestrained service composition as they develop applications in the space. This could be dangerous. Imagine a composed service of an automatic door opener with a strike-push mechanism getting invoked 50,000 times/s due to an infinite loop bug in the program. This will most likely damage the opener mechanism by causing it to be jammed or burning out. It may even result in a fire hazard. It may, as an end result, lead to the door staying open, which is unsafe and unsecure. Another example is a high-volume speaker going off in the middle of the night, also due to a programming bug. This could be devastating to a sleeping, frail elderly person with a weak heart. But even if the programming of an application is supposedly error and goof free, correctly programmed applications may interfere with each other or conflict in their goals, which is another ill-effect of the unrestrained service composition. By replacing carefully integrated elements with programmable services, we allow for all possibilities, including those that do not make sense or could lead to dangerous or harmful situations. **<AU: Please check whether replacing "by" with "with" in the preceding sentence conveys the intended meaning.>** We also open the door for programming errors and bugs to jeopardize the safety of the user, the space, or its devices. Surely, we need the elegance, flexibility, and power of SODA but we also need to safeguard against monkeying around, causing unpredictable or undesired consequences in smart spaces.

THE ANT AND THE ELEPHANT

Second comes the ant and the elephant. Presenting elements considered "fragile and weak" in a smart space as standard services tempts indiscriminate use beyond the service's capacity for duty. Fatiguing fragile services could very likely lead to major reliability, availability, and dependability problems. Dealing with and equating a pinhead device with a Dell server is obviously problematic, given the contrast in the mean-time-to-failure and the operational availability of each. This means applications may misbehave or themselves become unavailable. Once again, we wish to keep the advantages of SODA, but at the same time we should remove the mask and ensure that SODA is somehow aware and supportive to the varying power of its elements.

TOWARDS SAFER SMART SPACES

Safety research and development in smart spaces has been scant, perhaps because of market hesitation and the sluggish pace by which smart space technology has been entering the marketplace. Yet one may argue that had

safety been considered and built in from the start, perhaps market confidence would have been gained and adoption would have been accelerated for smart space products and services. At any rate, it is time now to avoid becoming a "Giraffe" and burying our heads in the sand without an adequate approach to this important problem. **<AU: Should "giraffe" be changed to "ostrich"?>** In the remainder of this column I will highlight key progress made in this area, which I stress is inadequate, with more concentrated efforts needed by the research community.

To cope with the ant and the elephant dilemma under SODA, compensating "sentience abstractions" defined over other services has been suggested. **<AU: Please check whether the preceding edited sentence conveys the intended meaning.>** The use of virtual sensors[5] is an example in which multiple replicas of the fragile elements in the space are employed, leading to redundant services grouped through quorum-based events to mask failures and extend the reliability and availability of these elements. Another sentience abstraction that has been introduced for the same reason is phenomena clouds,[6] which are more appropriate than virtual sensors for sensor grids where detecting user-sensor collision is of interest (such as an instrumented smart floor or smart bed). As shown in Figure 1, a phenomena condition is defined, along with transition rules, that clearly labels sensor nodes as idle, potential candidate, candidate, or tracking. **<AU: Kindly check whether the citation of Figure 1 is appropriate.>** Satisfying the phenomena condition makes a sensor node tracking. The phenomena may grow or shrink as it develops and a threshold phenomena diameter may be required before it is determined to have happened. **<AU: Please confirm changes to avoid repetition of "phenomena" in this sentence.>** Phenomena clouds have been used successfully in the Gator Tech Smart House[7] to sense user walking behavior much more reliably than by directly reading unreliable smart floor pressure sensors.

Empowering the ant through sentience abstractions as shown above seems to be a reasonable approach to back SODA's uniform service promise that all services are created equal. However, this would require building runtime systems for smart spaces that are able to support such abstractions simultaneously. For instance, raw data, events, contexts, phenomena clouds, activities, and behavior must all be supported so the programmers may pick the suitable one(s) in their applications.

To cope with the monkey, several approaches have been proposed, but much remains to be done. Chen et al[8] argued that by ensuring safety piecemeal, that is, separately ensuring safety of the physical space, the users, and the devices (the IoT), the overall safety of the space will be hugely improved. In this view, IoT device description language (DDL) was proposed[9] to enable device vendors to express, in human- and machine-readable form, use constraints of their smart space product. Such constraints could be anything ranging from general constraints that guard against device energy drains to device-specific constraints such as the strike-push door opener mechanism we discussed earlier. For instance, in the latter case, a use constraint could be that the mean time between two successive uses must be at least 2 s, which would perfectly guard against the 50,000/s invocation bug example mentioned earlier. Similar to the sentience abstraction idea, the DDL idea also has runtime implication. It requires that such constraints be runtime-registered exception handlers, which gives us clear cues to what a smart space runtime system should include. The DDL idea and the use of IoT constraints is not new, even though its application to digital smart space is. It is similar to how electric bumper cars in amusement parks function, where young riders are given the illusion they can turn the wheels infinitely and go flat out on the peddle; in reality, the constraints kick in at runtime, limiting the steering as well as the maximum speed.

Another approach to restraining SODA is to require so-called "safety

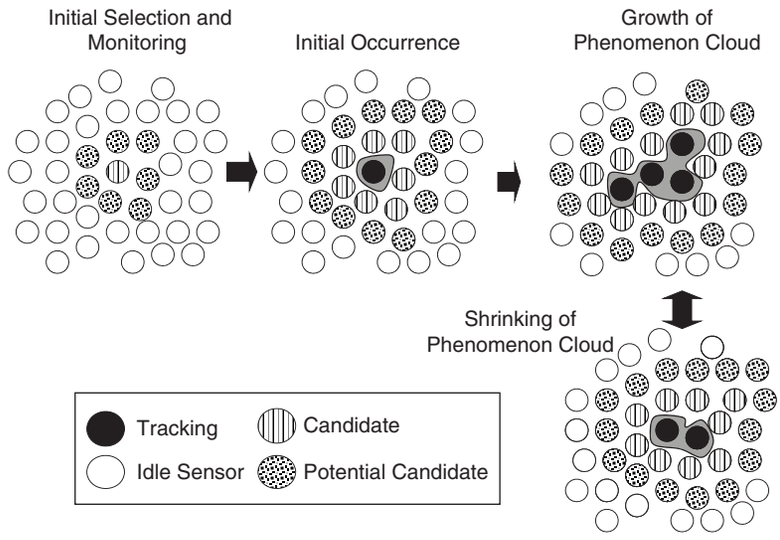

FIGURE 1. A phenomena cloud. **<AU: Kindly check whether the caption is appropriate.>**

containers" to be used by the programmers as they develop their applications, much like database transactions are required to ensure the proper handling of a database. IoT transactions (IoTransx) **<AU: Please check that "IoTransx" is spelled out correctly>** is a radical idea that has been proposed by Chen et al[8] where the smart space is modeled as a highly dynamic database. Unlike traditional databases and database management systems that take a "clean room approach" and cherish atomicity, consistency, isolation, and durability (collectively known as ACID), smart space databases take a "dirty room approach," where imperfection and the unattainability of full control and guarantees are the new normal and the sought goals. IoTransx aims to detect conflicting or interfering SODA services belonging to applications developed separately by independent developers. Similar properties to ACID are utilized by IoTransx except that consistency is replaced by the concept of integrity. Integrity is achieved through a context lock protocol and the compensating transaction to be used if the context lock started to significantly slip. The kind of locks used by IoTransx are ones that tolerate minor violations of the locking semantics in much the same way we would consider a slightly dripping water tap to be turned off. By developing all applications using such transactional programming constructs, the smart space runtime, again, would play a key role in ensuring the safety of the space despite conflicting applications.

Gadget app stores have started to appear, creating a similar ecosystem to the Apple apps store and Google Play. Smart spaces, therefore, have started to become a tangible market with pioneer products and services available to the consumer. Examples include most automobile brands, which today offer owners cloud accounts for purchased new vehicles to download after-sale services and apps. Samsung's line of smart home products is also being marketed through an app store named SmartThings,[10] which offers IoT devices for the home in addition to applications that are ready to download (called Scenes), providing more exciting applications out of the same set of gadgets purchased by the consumer.

To take smart homes to the next level of utility and usefulness, well beyond the simple applications of turning lights on and off as a motion sensor detects entrance to the living room, more research is needed to fully understand how to create safe applications with substantial actuations in smart spaces. Smart homes for active and healthy aging or supportive of frailty, dementia, and other conditions, just to give a few examples, require the utmost level of caution and safety as we develop their applications. To encourage more research in the areas of fault tolerance, robustness, and safety, funding agencies also need to bring these issues back into focus and highlight them in their programs. Such emphases were in place 15–20 years ago, but perhaps data science and artificial intelligence has overshadowed their importance.

REFERENCES

1. S. Helal and S. Tarkoma, "Smart spaces," *IEEE Pervasive Comput.*, vol. 14, no. 2, pp. 22–23, 2015. doi: 10.1109/MPRV.2015.40.
2. S. Helal, "Programming pervasive spaces," *IEEE Pervasive Comput.*, vol. 4, no. 1, pp. 84–87, 2005. doi: 10.1109/MPRV.2005.22.
3. S. de Deugd, R. Carroll, K. Kelly, B. Millett, and J. Ricker, "SODA: Service oriented device architecture," *IEEE Pervasive Comput.*, vol. 5, no. 3, pp. 94–96, 2006. doi: 10.1109/MPRV.2006.59.
4. J. King, R. Bose, S. Pickles, A. Helal, S. Vanderploeg, and J. Russo, "Atlas: A service-oriented sensor platform," in *Proc. 1st IEEE Int. Workshop on Practical Issues Building Sensor Network Applications (SenseApp)*, 2006. **<AU: Please provide the page range.>**
5. R. Bose, A. Helal, V. Sivakumar, and S. Lim, "Virtual sensors for service oriented intelligent environments," in *Proc. IASTED Int. Conf.: Advances Computer Science and Technology*, Apr. 2007, pp. 165–170.
6. M. Thai, R. Tiwari, R. Bose, and A. Helal, "On detection and tracking of variant phenomena clouds," *ACM Trans. Sens. Netw.*, vol. 10, no. 2, Jan. 2014. doi: 10.1145/253052.
7. S. Helal, W. Mann, H. El-Zabadani, J. King, Y. Kaddoura, and E. Jansen, "The gator tech smart house: A programmable pervasive space," *Computer*, vol. 38, no. 3, pp. 50–60, Mar. 2005. doi: 10.1109/MC.2005.107.
8. C. Chen and A. Helal, "System-wide support for safety in pervasive spaces," *J. Ambient Intell. Hum. Comput.*, vol. 3, no. 2, pp. 113–123, 2012. doi: 10.1007/s12652-011-0078-7.
9. C. Chen and A. Helal, "Device integration in SODA using the device description language," in *Proc. 2009 9th Annu. Int. Symp. Applications and Internet*, Bellevue, WA, pp. 100–106. doi: 10.1109/SAINT.2009.24.
10. "SmartThings: More smart devices, one smart app," Samsung, Seoul, Korea. [Online]. Available: https://www.samsung.com/global/galaxy/apps/smartthings **<AU: Please provide the year or date of access if possible. Also, please confirm the inserted title and the name and location of the associated organization.>**

SUMI HELAL is a professor and chair in Digital Health at Lancaster University, Lancashire, England. He is a Fellow of the IEEE, AAAS, and IET. Contact him at sumi.helal@ieee.org.

Several efforts began almost two decades ago to find successful models for “programming pervasive spaces.”

Soon after celebrating the successful SODA milestone, it became evident we just created an unrestrained programming model that lacks adequate safety guards.

We realized that SODA was too powerful to be safe.

Even if the programming of an application is supposedly error and goof free, correctly programmed applications may interfere with each other or conflict in their goals.

By replacing carefully integrated elements with programmable services, we allow for all possibilities.

Fatiguing fragile services could very likely lead to major reliability, availability, and dependability problems.

Safety research and development in smart spaces has been scant.

More research is needed to fully understand how to create safe applications with substantial actuations in smart spaces.