Editor: Gustavo Rossi • gustavo@lifia.info.unlp.edu.ar

Task Automation Services: Automation for the Masses

Miguel Coronado and Carlos A. Iglesias • *Universidad Politécnica de Madrid*

While outlining the features and components of task automation services (TASs), here the authors propose a generic architecture that supports current challenges. They also provide a survey comparison of existing platforms and discuss their evolution and future tendencies.

Task automation permeates our daily lives, from the weather forecast that appears when the alarm clock rings, to the smartphone notification that pops up every time we receive an incoming email. These automations orchestrate gadgets, Internet services, and apps in ways that make our life easier.[1] We're so accustomed to task automations that sometimes it's hard to identify them, and even harder to recall that some years ago we performed those tasks manually.

While these predefined task automations are spreading across the Web, a new user-centered fully customizable approach is beating them all, the so-called *task automation service* (TAS). TASs are typically Web platforms or smartphone apps that provide a visual programming environment, where non-technical users can seamlessly create and manage their own personal automations.[2] The automation in these services takes the form of Event-Condition-Action (ECA) rules that execute an action upon a certain triggering event, such as "when *triggering-event* then do *action*." For example, the alarm clock and incoming email would be the triggers, whereas querying the weather forecast and displaying a notification are the respective actions.

Some TASs have become mainstream, such as Ifttt (http://ifttt.com)[3] or AutomateIt (http://automateitapp.com). In 2014, Ifttt reported more than 14 million Web tasks created by users. AutomateIt, an Android application, has more than 500,000 users. There are three success factors that explain their growing adoption. The first is usability; TASs provide a simple-yet-powerful intuitive interface for programming task automations. Hence, users experience almost no learning curve when they start using them. The second factor is customizability; TASs let users program the automations they need. Although simple, automations are powerful improvements to users' daily lives. Giving users the capability to create their own rules awakens a sense of control and immediacy – they get the automations they need when they need them. The third factor is integration with existing Internet services. Users can automate tasks that access the Internet services they already use and are familiar with.

Given their novelty, here we shed light on TASs, presenting and exemplifying their main characteristics. Motivated by their relevance and penetration in the market, we survey some of the most prominent TASs and classify them according to different dimensions. We also define a reference TAS architecture that identifies TASs' key elements and interactions. This architecture provides a common vocabulary and serves as a reference that can accelerate the development, adoption, and evolution of TASs.

Scenario and Challenges

To better understand what a TAS might look like, consider the following scenario. Sarah uses a TAS every day, so she has defined a set of useful automations. Some automations notify her when something relevant to her occurs, such as "when I'm mentioned on Twitter, send me an email." Others save her the bore of repeating a simple task – for example, "when I'm tagged in a Facebook picture, save it to my Dropbox" or "convert incoming invoice emails to PDF and store them in my Evernote." Furthermore, Sarah also uses the smartphone app provided by the TAS. Once

 1089-7801/16/$33.00

installed, the TAS can access several resources from her smartphone, so she can set up rules involving incoming calls, the camera, Bluetooth, or GPS among others. Rules such as "when my smartphone's battery level is under 10 percent, text my parents" or "when I get to work, lower the volume of my ringtone" take advantage of those capabilities. Moreover, the TAS feature Sarah enjoys most is the discovery of compatible services around her — using smartphone communication capabilities such as Bluetooth. This feature can automatically integrate her smart TV or her home automation lighting system with the TAS, allowing her to set rules for home automation, such as "when my alarm clock rings, switch on the bedroom lights."

This brief journey with Sarah illustrates how services and sensors can be connected by means of automation rules defined with a single TAS. It aims to give a clear view of a TAS's functioning and main features, and it also outlines some challenges, such as embracing smartphone resources or auto discovery of services; we address these challenges in the following sections. Next we discuss the elements this scenario introduced.

TAS Components: Channels and Execution Profiles

Our scenario combines events from Internet services, Sarah's smartphone, and connected devices. These services and devices are managed by channels. We define *channels* as abstractions for receiving events or emitting actions to Internet services (Web channels) and connected devices (device channels). Channels should be registered in a channel directory service provided by the TAS. In this way, users can activate available channels when programming automations.

Web Channels

Many TASs rely on third-party Internet services to supply a pack of useful, popular, user-tested channels. Hence, users benefit from using TASs to manage the services they're already subscribed to (such as Evernote or Gmail). As a result, TASs provide users with a new layer of control to manage their services and they aren't required to migrate.

By analyzing the behavior of Web channels, we identify three characteristic dimensions. Consider a user who wants to define a new automation rule. First of all, that user needs to grant the TAS access to the Internet service, usually by providing access credentials — this is what the *privacy paradigm* defines. In addition, some channels must be configured. This is the case for the weather forecast channel, in which users must provide a location for the forecast — this is defined within the *configuration paradigm*. Finally, channels might behave differently triggering the rule or being the consequence that takes place — that is, they could generate events, provide actions, or both — this is the *input/output (I/O) paradigm*.

From the privacy paradigm's point of view, channels may be public or private. To activate a channel and let the platform act on behalf of users, users must grant access to the service. In our former example, Sarah had to allow the TAS to access her Dropbox account and email inbox to manage her files and emails. When this authentication is required, the channel is private. The privacy paradigm defines who will have access to events and actions provided by the channel. Information regarding private channels is for the user's eyes only, and it's tailored to the user. On the contrary, channels that don't ask for authentication are public channels, and every user gets the same information when using them. This is the case with newsfeeds or weather forecast channels. The privacy paradigm also covers *private group channels*, where every group member receives all the events the channel generates. These channels are common in scenarios such as home automation, where family members are likely to share home channels.

The configuration paradigm defines the setup needed to activate a channel — apart from authentication. Public channels usually require configuration for the sake of a better user experience and also as a matter of efficiency. For instance, in our example to activate a weather channel, Sarah provided her home location. Hence, she receives weather events related to that location. In general, private channels don't require configuration because they're already tailored to the user.

Finally, when activated, channels might generate events, provide actions, or both. This is what the I/O paradigm defines. Events are changes in the service's state, such as a new email in Sarah's inbox. On the other hand, channels might also offer action capabilities, such as switching on the bedroom light. The I/O paradigm also covers *pipe channels*: those that process the input to generate a different output that can be wired to another channel. For instance, the PDF converter channel that saves the content of Sarah's email into a PDF file is then connected to the Evernote channel to store the file.

From an integration perspective, most of the efforts in offering a new Web channel are related to implementing the protocol to communicate with the Internet service behind the channel. This is the TAS administrators' duty, which depends on the availability of an API for the Internet service. The authorization process involved in accessing the Internet service API determines the privacy paradigm. Besides communication with the service, the TAS administrators define what events and actions will be offered as part of the I/O paradigm as well as the configuration paradigm.

Device Channels

In comparison to Web channels, device channels manage data from the connected devices they oversee. In Sarah's scenario, her home automation lighting system and the smart TV provide device channels to control all the switches of her home and her smart TV, respectively.

From a behavioral approach, device channels respond to the same three dimensions analyzed for Web channels. In addition, device channels implement two additional dimensions: the communication and discovery paradigms. The *communication paradigm* defines how the communication between the devices and the channel will be carried out: wired, wireless, through the Internet, and so on. It also defines on what conditions the channel is available. As opposed to Web channels, which can be accessed from all around the world through an Internet connection, access to device channels may depend on local aspects. These aspects are part of the communication paradigm. For instance, when using wireless communication, availability is subject to the distance between devices—that is, being under the coverage area or not. Finally, device channels could announce themselves so that they're available for automation, as in Sarah's scenario. This is what the *discovery paradigm* defines. It provides standard operations and APIs to enable self-identification of devices, capabilities discovery, and access to device data using predefined message structures. As a result, the system provides a plug-and-play capability.

From the implementation's point of view, each TAS administrator decides which communication protocols will be supported in the platform. Each protocol has its own restrictions about range, power consumption, number of connected devices, and so on. They also provide mechanisms such as security and device discovery. To communicate two devices, they need to support the same protocol. However, this shouldn't be an issue, as a device may implement several protocols. In fact, many devices are compatible with the most widespread protocols, such as WiFi, ZigBee, Z-Wave, and even Bluetooth.[4]

Rule Execution Profiles

Next, we describe automation rules, which provide the logic to connect channels. As previously stated, TAS automations address simple ECA rules.[5] The rule's event and action might come from the same or different channels. However, more complex rules could be devised: *multi-action rules* can execute several actions in parallel when the rule is executed; *multi-event rules* are triggered by a combination of events; and *chain rules* execute a list of actions in sequential order, so the output of an action might be used to trigger the next rule. Complex rules require the TAS to support additional features. For instance, multi-event rules require complex event processing (CEP) support to evaluate complex patterns of events, and chain rules make use of pipe channels, so they must be supported by the TAS.

Group rules are particular kinds of rules that involve several users and are susceptible to collisions with other rules. For instance, Sarah's smart lighting system is a shared resource managed by a group channel. If Sarah defines a rule to switch off the corridor lights while she's asleep, and her flatmate had a rule that turns them on when the alarm clock rings, both rules collide. It's easy to get to a point where the TAS can't determine whether the lights should be on or off. As a rule of thumb, rules that include group channels are group rules.

The *rule-execution profile* defines where the execution of the rule takes place. Rules can be executed according to different execution profiles to increase performance and enable offline rule execution. In Sarah's scenario, she uses a TAS hosted in the Web, but we can imagine other scenarios where a smartphone or a set-top-box performs the automation. Rule execution can be accomplished according to three different execution profiles: entirely on the Web, on the mobile client, and mixed execution. A *Web-driven execution profile* centralizes the execution on the server, allowing the existence of lightweight clients at the cost of requiring an Internet connection. A *mobile-driven execution profile* is orchestrated on the client (such as a smartphone or set-top-box), allowing offline rule execution when only local device channels are involved. A *mixed-execution profile* takes the advantages of both profiles. It can shift the execution to the client or to the Web, which also reduces the communication payload between the client and server.

A Reference TAS Architecture

Now that we've described the main components of TASs, here we define TAS as a service that lets users create automation rules that connect channels using a visual editor. A reference TAS architecture must incorporate the features and components previously described (channels, rules, and execution profiles) and also provide support for some of the challenges presented in Sarah's scenario. The following are the architecture requirements:

- provide a visual rule editor for creating rules;
- include both Web and device channels;
- feature channel discovery, providing adapters for connected devices so that they're reachable directly by the platform;
- enable multi-event, multi-action, and chain rules;
- manage group channels and group rules;
- detect collisions with rules that involve group channels; and
- support a mixed-execution profile.

The architecture must provide a visual automation rule editor that users could access to create their automations, but also to activate channels according to the privacy paradigm. To provide access to Web and device channels, the architecture must provide the logic needed to connect with Internet services and connected devices. Moreover, the TAS must be notified without delay when the channels generate an event, and it must be able to send

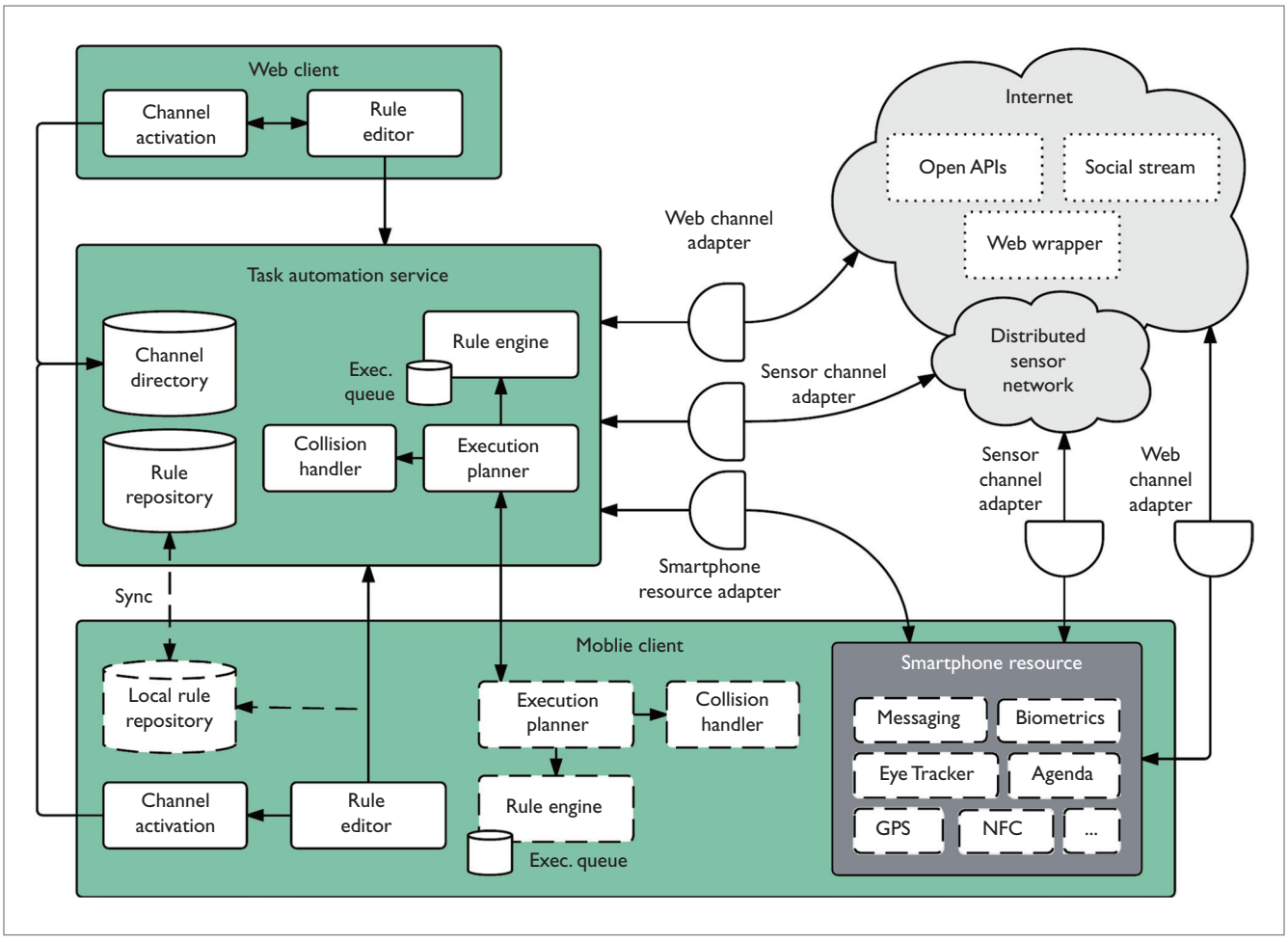

Figure 1. Reference task automation service (TAS) architecture general diagram. Elements in dashed lines are optional in some implementations.

actions. It also must be able to discover channels according to the device channels' discovery paradigm.

Advanced rule features such as multi-action rules and chained rules don't imply additional requirements, because they can be translated in a set of simpler rules. Multi-event rules involve temporal reasoning of events, and so the TAS requires CEP facilities.[6] Nevertheless, a tradeoff among usability for end users and expressivity should be reached for multi-event rules. Group rules require group channels' support and rule-collision handling. To handle collisions, the architecture must be able to first detect them, and then act when the collision occurs and prevent any unwanted effects.

Supporting mixed-execution profiles requires some additional logic to coordinate server and device-automation rule engines while they orchestrate rule execution. This logic is also needed to guarantee that the information about the user and the rules are synchronized on the device and server.

Once the requirements to support these novel features are clear, we introduce a reference architecture that fulfills them (see Figure 1).

Rules may be created using an editor on a Web client or a mobile client. They're stored in a central rule repository on the TAS server. However, because those rules can be executed on the client according to the mobile-driven execution profile, they're synchronized with a local rule repository for offline access. Mobile and Web clients also let users activate channels. Once a channel is activated, it's saved in the channel directory together with the authorization credentials. These credentials are used by the adapters to access the channels.

Adapters provide a uniform access to all kinds of devices. They're responsible for notifying the TAS of incoming events, commanding the execution of actions, and taking charge of channel authentication. Adapters can be implemented following a publish-subscribe[7] or polling strategy to get notified of device events, depending on their nature. The implementation of adapters can be done by the TAS

administrator or by third parties if the TAS provides an adapter software development kit (SDK). Adapters to sensor channels provide two different paths: access through a Web protocol, or direct access using access protocols such as ZigBee, Z-wave, Bluetooth, or WiFi. They usually expose a set of sensors, such as a sensor network, but single device channels are also feasible.

To support a mixed-execution profile, modules involved in rule execution and channel access must work in coordination. This is the case of the automation rule engine responsible for executing rules and managing rule lifecycles within the execution query.

To support a mixed-execution profile, modules involved in rule execution and channel access must work in coordination.

Rule execution consists in fetching the incoming triggering event, extracting the arguments, and using those parameters to request the action execution. The execution planner guides the orchestration from a higher level according to the active rule-execution profile. It manages the channels' state (within the channel directory), tracking when channels are down and new channels are discovered. For instance, when Sarah comes home, the smart TV channel is discovered and added to the channel directory. In turn, when she leaves home, the channel will be down because it's out of range.

Finally, the collision handler analyzes the rules in the repository, searching for possible collisions. Some patterns of collision are easy to detect — for example, the simplest collision consists of two rules with the same triggering event that try to execute two opposite actions. First, the collision handler considers which users are connected currently to each channel (as registered in the channel directory), because the rules of two users can only collide if both are connected to the same channel. Recall the example where Sarah wants to have the corridor lights off while she's asleep, and her roommate set a rule to switch them on when the alarm clock rings. If Sarah's roommate is not at home, there's no possibility of collision because the channel isn't active for Sarah's roommate. The execution planner consults the collision handler before executing a rule, and if that rule collides with other rules, its execution is skipped.

Analysis of Current TAS Platforms

To determine which of the features discussed are supported by state-of-the-art task automation, we analyzed Web platforms for a general audience (Ifttt), Web platforms for businesses and enterprises (Zapier, CloudWork, elastic.io, and itDuzzit), a Web platform for cloud storage synchronization (Wappwolf), mobile apps (Tasker, Atooma, AutomateIt, and on{x}), and smart home systems (WigWag and Webee). Table 1 shows a summary of the results; the complete report is available at http://bit.ly/TASStudy.

As expected, Web channel support is much larger than device channel support. Although the studied apps manage the resources in the smartphone, only home automation-related TASs provide device channels that connect directly to devices. Pipe channels are only fully supported by elastic.io. It's a powerful concept, and it integrates perfectly with the interface; however, it's barely used. It's worth mentioning that home automation TASs feature channel discovery and group channels. Unfortunately, their support is still extremely limited.

Regarding different types of rules, few TASs support multi-event rules' complexity in comparison to simpler rules. Multi-action rules have wider support, but still many TAS managers don't include them in their platform in order to keep rule editors simple. In the end, a user can achieve the same functionality by implementing a rule for each action in the multi-action rule. Elastic.io features chain rules, and WigWag and Webee (which support group channels) feature group rules; however, none of them handle collisions in group rules. Finally, some of the TASs provide their users with a pack of predefined rules that are useful for previous users, because they act as a shortcut for creating those automations.

At the time we performed this study, none of these platforms supported a mixed-execution profile. Thus, we split them into those with a Web-driven execution profile (Ifttt, Zapier, CloudWork, elastic.io, itDuzzit, and Wappwolf) and those with a device-driven execution profile (Tasker, Atooma, AutomateIt, on{x}, WigWag, and Webee). By their nature, platforms with a Web-driven execution profile have more limited access to device channels than those executed on the smartphone. The latter has access to all the smartphone resources. For that reason, Ifttt has already released a smartphone app that grants Ifttt server access to smartphone resources. In turn, Zapier includes Tasker as a channel that effectively grants access to those channels, too.

Several TASs offer advanced features for users with programming skills to set up automations using a programming

Table 1. Comparison of TASs.

		Web						Smartphone				Home	
		Ifttt	Zapier	CloudWork	Elastic.io	ItDuzzit	Wappwolf	On{x}	Tasker	Atooma	AutomateIt	WigWag	Webee
Channels	Web channel support	✓	✓	✓	✓	✓	Few	Few	✓	✓	✓	✓	✓
	Device channel support	Few	×	×	×	×	×	✓	✓	✓	✓	✓	✓
	Smartphone resources as channels	✓	×	×	×	×	×	✓	✓	✓	✓	×	×
	Public channels support	✓	×	×	✓	✓	×	✓	×	×	×	×	×
	Pipe channel support	×	×	×	✓	×	Few	Few	×	×	×	×	×
	Group channel support	×	×	×	×	×	×	×	×	×	×	Few	×
	Device channel discovery	×	×	×	×	×	×	×	×	×	×	✓	Few
Rules	Multi-event rules	×	×	×	×	×	×	✓	✓	×	×	✓	×
	Multi-action rules	×	×	×	✓	×	×	✓	✓	×	✓	✓	✓
	Chain rules	×	×	×	✓	×	Few	Few	×	×	×	×	×
	Group rules	×	×	×	×	×	×	×	×	×	×	Few	×
	Collision handling	×	×	×	×	×	×	×	×	×	×	×	×
	Predefined common rules	×	×	✓	×	✓	✓	×	×	×	✓	✓	✓
	Rule execution profile	WD	WD	WD	WD	WD	WD	DD	DD	DD	DD	DD	DD
TAS	Visual rule editor	✓	✓	×	✓	✓	✓	×	✓	✓	✓	✓	✓
	Provides API	×	✓	×	✓	×	×	×	×	×	×	×	×
	Programming language	×	×	×	✓	×	×	✓	Few	×	✓	✓	✓

* ✓ = supported; × = not supported; Few = few support; WD = Web-driven execution profile; and DD = device-driven execution profile.

language. This is the case with on{x} (JavaScript), AutomateIt (Bash), elastic.io (JavaScript), ItDuzzit (proprietary), WigWag (Arduino/Raspberry/JavaScript), and Webee (Boss). Moreover, Zapier, elastic.io, Tasker, and Webee provide an API for developing channels (some call them plug-ins).

As TASs gain in popularity and existing TASs increase their number of users, new services appear and compete. As a novel domain, task automation opens the door to several opportunities – yet many challenges remain.

First, a certain degree of standardization is required. Because each TAS wages war on its own, Internet service developers might find their services available as channels in some TASs, but not in all of them. Moreover, when users define rules within the scope of a TAS, these rules can't be exported to others or shared. Second, TASs should evolve into a mixed execution profile that's able to interact with Web and device channels. Third, TASs that include device channels to manage shared devices (such as a smart TV or a smart lighting system) should allow group rules and consider the mechanism for detecting and handling collisions. This is quite challenging and will require complex algorithms. Finally, TASs should include mechanisms for discovering device channels so that their configuration is almost transparent to the end user.

Acknowledgments

This work was supported by funds from the Spanish Ministry of Economy and Competitiveness (project CALISTA, TEC2012-32457), and by the Autonomous Region of Madrid through program MOSI-AGIL-CM (grant P2013/ICE-3019, co-funded by EU Structural Funds FSE and FEDER).

References

1. K. Parks and D. Watkins, *How to Automate Your Way to Freedom*, ebook, 2012; www.amazon.ca/How-Automate-Your-Way-Freedom-ebook/dp/B0087UNI1S.
2. A.R. Meisel, *Optimize, Automate, and Outsource Everything In Your Life: How to Make*

Email, IFTTT, and Virtual Assistants Your Ultimate Productivity Weapons, CreateSpace Independent Publishing Platform, 2014.
3. A. Martinez et al., *The Ultimate IFTTT Guide: Use the Web's Most Powerful Tool Like a Pro*, ebook, 2013; www.amazon.com/The-Ultimate-IFTTT-Guide-Powerful-ebook/dp/B00FK1X1YQ.
4. H. Labiod et al., *Wi-Fi Bluetooth, Zigbee and WiMax*, Springer Science and Business Media, 2007.
5. W. Beer et al., "Modeling Context-Aware Behavior by Interpreted ECA Rules," *Euro-Par 2003 Parallel Processing*, Springer, 2003, pp 1064–1073.
6. M. Eckert et al., "A CEP Babelfish: Languages for Complex Event Processing and Querying Surveyed," *Reasoning in Event-Based Distributed Systems*, Springer, 2011, pp 47–70.
7. P.T. Eugster et al., "The Many Faces of Publish/Subscribe," *ACM Computing Surveys*, vol. 35, no. 2, 2003, pp 114-131.

Miguel Coronado is a PhD student in the School of Telecommunications Engineering at the Technical University of Madrid. His research interests include the Semantic Web, linked data, personal assistants, and the Internet of Things. Miguel has an MSc in telecommunications engineering from the Technical University of Madrid. Contact him at miguelcb@dit.upm.es.

Carlos A. Iglesias is an associate professor and head of the Intelligent Systems Group in the Telecommunications Engineering School at the Universidad Politécnica de Madrid. His research interests include Web technologies, agent systems, social intelligent systems, the Internet of Things, and big linked data analytics. Iglesias has a PhD in telematics from the Technical University of Madrid. Contact him at cif@dit.upm.es.

cn *Selected CS articles and columns are also available for free at http://ComputingNow.computer.org.*