Task as a Service

Extending the Cloud From an *App Development Platform* into a *Tasking Platform*

A Position Paper

Joanna W. Ng
IBM Canada Software Laboratory, CAS Research
jwng@ca.ibm.com

Abstract

This paper coins the term "***Tasking***" and defines a "***Tasking Conceptual Model***" as a software engineering approach. In this approach, instead of producing Apps as a runtime artifact *whole* for its users, software developers produce intermediary artifacts, as accessible controls *parts* for average users. Using these controls, users are enabled to *construct their own tasks autonomously* by using resources of their own choice from across the cloud. I propose using *web tasking over the cloud* as a tasking approach to provide (1) *average users* consistent and universal tasking experience across the cloud despite of the resource diversity; and also, to provide (2) *developers* prescriptive, standard-ready software engineering steps to produce and contribute web-tasking resources. This proposed approach also provides (3) built-in *interoperability*. Users can use web-tasking resources produced by independent parties and expect that they can interoperate seamlessly together. I also propose *a new cloud layer for users* on top of the existing cloud layers *for developers* to create Apps. I coin this new cloud layer for end users "Task as a Service" (TaaS).

Keywords: *Web services, web tasking, RESTful Architecture, Web Automation, Web Agents, Cloud, Cloud Infrastructure, Software as a Service, SaaS*

1 Introduction

Cloud provides a rich and efficient environment for software developers to develop, deploy and run Apps for their users [1]. Apps are today's mechanism for users to perform specific tasks, constrained by how Apps-programmers program them. App-developers control what cloud resources to use and how interaction paths are put together. While Apps are widely used and highly popular, Apps have their perils. *Firstly*, the exponential growth in number is overwhelming. *Secondly*, App-users have no control. Apps are meant to be used by generic users. They are not meant for addressing personalized and situational requirements. There is a gap for end users in the current state of software engineering.

Take car engineering as an analogy. If only car engineers have enough skills and knowledge to drive a car, then the world has no car for the general public. But because car engineers design *accessible controls* for their users (e.g. wheels, ignition device etc.) by abstracting away the complexity of car engineering, average users can drive cars by themselves, independent of car engineers, without any cognizance of car engineering.

Applying this *driving* analogy in *tasking* from the end users' perspective, how can complexity in software engineering be abstracted into simplified controls that average users can use to control task for themselves, independent of software engineers, and without any cognizance of software engineering? What are the forms of *tasking controls*, equivalent to wheels and ignition in driving, that is accessible to average users? Pursuing answers and solutions to the above question is what motivates the work and contribution of this paper.

There are quite a number of recent researches in this problem space that can be grouped by their approaches. Firstly, there is the *end-user programming* approach [2, 9]. Secondly, there are mash-ups [2, 10]. Thirdly there is the visual programming approach using *wiring* [5]. Fourthly, there are early implementations of tasking approach, using control-metaphor without requiring any cognizance of software engineering nor programming from the end users. IFTTT [3] and Zapier [4] are two very popular examples.

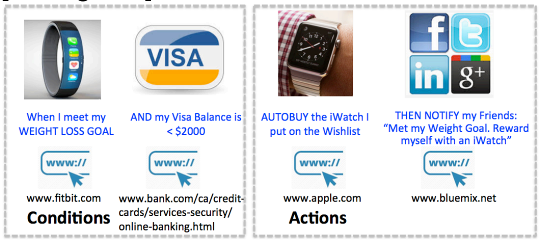
In this paper, I establish *a tasking conceptual model*. I also establish the importance of an open tasking model and describe the features that make a tasking model open. I *compare* existing tasking approaches to that of web tasking [7]. I further propose to extend tasking into the cloud space, adding "*Task as a Service*" (TaaS) as a cloud layer of abstractions for user tasking on the cloud.

The remainder of this paper is organized as follows. Section II describes the scenarios. Section III discusses related work. Section IV establishes a tasking conceptual model. Section V proposes Tasking as a Service as a solution. Section VI concludes the paper and proposes future work.

2 Scenario

This section highlights a few TaaS scenarios to illustrate the gaps between today's Apps and users' personalized and situational task requirements.

Imagine an online shopper wants to buy an iWatch based on the goals that he set for himself. A use case would be to specify two conditions: (1) weight loss goal of twenty pounds and (2) his visa spending for the month is less than two thousand dollars, then to specify the action sequence: (1) order of iWatch, (2) followed by notifying this friends with a message. The user wants to have control over the conditions and the action sequence of his task and *automate the triggering of its execution* once the specified conditions are met [See Figure 1].


When I meet my WEIGHT LOSS GOAL
AND my Visa Balance is < $2000
AUTOBUY the iWatch I put on the Wishlist
THEN NOTIFY my Friends: "Met my Weight Goal. Reward myself with an iWatch"
www://
www.fitbit.com
www.bank.com/ca/credit-cards/services-security/online-banking.html
www.apple.com
www.bluemix.net
Conditions
Actions
Figure 1: A TaaS Use Case

Such daily tasks have to be so easy to put together by average users that it can become a regular, day-to-day undertaking unassisted by IT.

Internet of Things (IoT) also adds to the validation of the need for TaaS. With fast increase of devices, users need to take over the control of IoT devices in order to set them up for their own purposes. For example: if smart-energy thermostat saves $500, notify family with a message, "well done, guys!" and auto order the iTV.

There are also many TaaS scenarios in the B2B space that can greatly improve the enterprise businesses operational efficiency by web task automation. For example: if a product item has less then ten left in the inventory and the price for replenishment is not greater than a certain amount, plus the supplier is in good standing, then automatically replenish through the cloud. Many similar scenarios can be derived in other domains such as healthcare, smarter cities etc. to illustrate the need of a tasking approach.

Here is a summary of a common set of characteristics that point to the need for tasking because Apps won't do:

- Users *have a need to take charge of the control of task conditions*, including:
  - The choice of resources
  - The specification of condition combinations
- Users *have a need to take charge of the control of task actions*, including:
  - The choice of resources
  - The specification of action sequence for execution
- Users want to *delegate the checking of condition fulfillment and the subsequent initiation of task execution*.
- Users have a requirement for *a light-weight, minimalist approach*. For example: Coding-deploy-run platform for Apps is completely inappropriate for emergency room doctors, who needs to set up unique patient's vital-sign combination of conditions for patient monitoring. Even though they have the capability to acquire programming skills, their on-site situations require them to use tools that are drastically simple and accessible in their ER setting.

3 Related Work

In this section, I summarize the related work categorized by their approaches.

3.1 End User Programming

There has been significant research in End-user software engineering (EUSE), distinguishing (i) *end-user* programming from *professional* programming; with an *intent-based* differentiation: End-user programming produces program for personal use. Professional programming produces program to be used by larger and more generic groups of users [9].

EUSE research takes various approaches ranging from (i) simplifying the acquisition of programming skills; to (ii) providing programming assistance by using metaphors in graphical interfaces, or by using abstracted representations to hide complex programming concepts, or by providing helper codes to avoid syntactical errors etc.[2,9].

Scratch from MIT [11] has taken EUSE to the furthest by providing a gamification style of programming as a stepping stone to computer programming for end users, leveraging interactive story characters, music and many other entertaining features [See Figure 2].

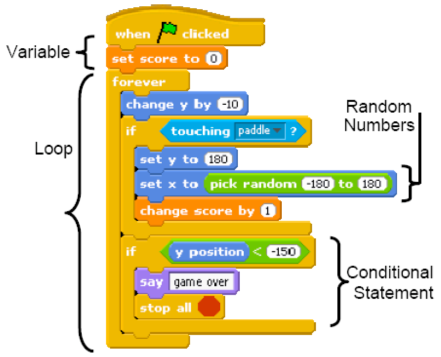


Figure 2: Scratch End User Programming by Gamification

Nevertheless, *programming is still primary metaphor*. In reality, there are many end users' day-to-day settings that programming environment is plainly inappropriate and inaccessible for the situation. Previous examples of emergency room situations for doctors, or on-site inventory checking and mobility requirement for B2B inventory tasks are some good examples.

3.2 Visual Programming

Visual programming refers to a set of interaction technique and visual notations for expressing programs. Elements of programming language such as loops; development environment such as containers or runtime libraries etc. are abstracted into graphical representations [9]. *Wiring* is the typical *end-user control* that users used to put them together to create the program, and help users to overcome the cognitive difficulties in programming.

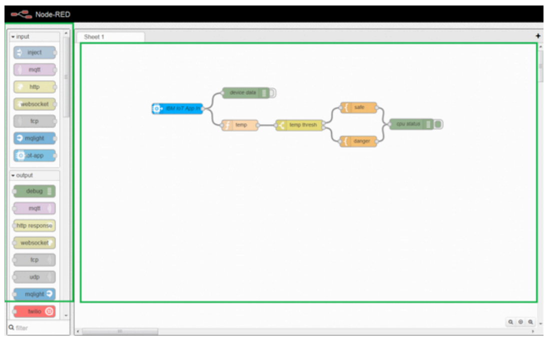


Figure 3: Node.Red Flow Editor as an example of visual programming to create apps

Node.Red is a typical example of visual programming through wiring, availing in IBM's Bluemix cloud platform [12]. While it is a much-simplified programming environment than today's IDEs, *programming is still the central metaphor*. The graphical representations expose technological constructs such as "WebSocket", "httpResponse", "Tcpip", "mqtt" etc., [See Figure 3]. End users have to acquire the associated technical knowledge to master it [5]. It has the same challenges of appropriateness and accessibility when used in real life, on-site scenarios as previously stated.

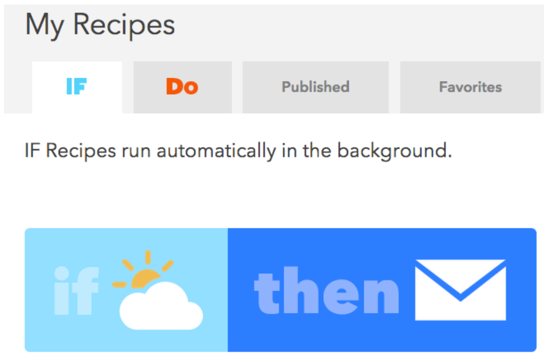
3.3 Mashups

Mashups is designed to combine existing web-based content and services to create new applications. This is a significant breakthrough in that it stops using programming as the central control metaphor like EUSE or visual programming. Instead, it breaks new ground by using a control-metaphor that is in the users' domain such as "spreadsheets" and "pipes" [10]. However, the preparation steps to setup the pipes and spreadsheets, including the cleansing of data crawling for the table, or customizing the 'operators' of these pipes, or customizing the operators required low-level technical skills such as data processing or programming. As a result, this mashup approach suffers adoption hurdles by average users in their every day situations.

3.4 Tasking

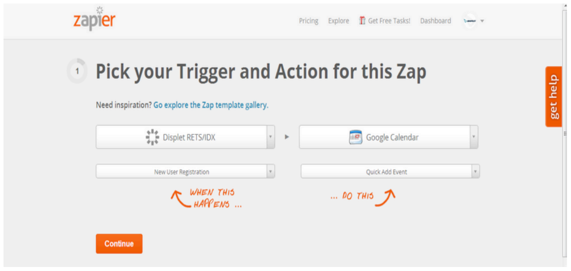
The recent flourishing of start-ups, such as IFTTT [3], Zapier [4] and many others, offer average users freedom to associate "condition" with "action". These offerings have become a trend too prominent to ignore. I coin this approach "***Tasking***", as distinctively different from end-user programming, visual programming and mashups.

IFTTT [See Figure 4] names it "Recipe", Zapier [See Figure 5] names it "ZAP". Tasking in IFTTT is proprietary and internal. ZAP's tasking is by vendor-specific scripting.



IF Recipes run automatically in the background.

Figure 4: IFTTT User Interface for "Recipe" [3]



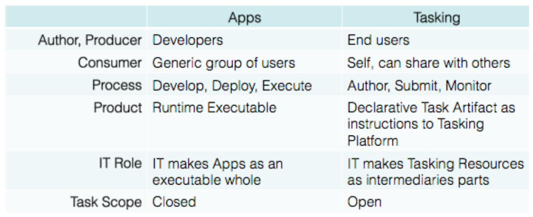
Pick your Trigger and Action for this Zap

Need inspiration? Go explore the Zap template gallery.

Figure 5: Zapier User Interface for "ZAP" [4]

4 A Tasking Conceptual Model

Figure 6 summarizes the difference between today's Apps and the Tasking approach. ***Tasking*** is the new paradigm that breaks entirely away from the programming metaphor yet not requires users to acquire any programming or technical skills.



	Apps	Tasking
Author, Producer	Developers	End users
Consumer	Generic group of users	Self, can share with others
Process	Develop, Deploy, Execute	Author, Submit, Monitor
Product	Runtime Executable	Declarative Task Artifact as instructions to Tasking Platform
IT Role	IT makes Apps as an executable whole	IT makes Tasking Resources as intermediaries parts
Task Scope	Closed	Open

Figure 6: Apps versus Tasking

This paper establishes a tasking conceptual model [See Figure 7] that has three logical components. Firstly, there is a "***Tasking Platform***" provided by a tasking vendor (such as Zapier). Secondly, there is a "***Tasking Resource Representation***". It is a resource model defined by the tasking vendor to prescribe how IT can take their entities from current Apps or curated services and transforms them into tasking resources for the tasking platform. Thirdly, there is a "***Tasking Control-Metaphor***". It is a control-model that users use to maneuver tasking resources from the tasking platform to create their own personalized tasks.

IFTTT and Zapier each have its tasking platform. Both tasking platforms adopt "Tasking Template" as the tasking control-metaphor. There is only one task pattern for the template, which is "Condition-then-Action".

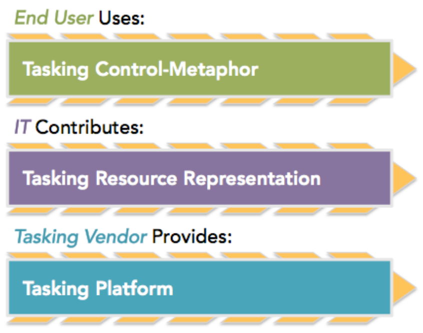


Figure 7: The Tasking Conceptual Model

Current implementations of the tasking approach have their limitations and challenges, reflecting its infancy. *Firstly*, current tasking platforms and the tasking resource representations are either proprietary or are built upon vendor-based scripting language. They are not open in architecture and therefore not a candidate to mature into widely adopted industry standards in the future. There is no open APIs available from these tasking platforms, making it impossible for third parties to build upon them. This inhibits open participation and contributions from developers. The consequence for prohibiting broad adoption is severe. *Secondly*, current tasking implementations lack a well thought out integration architecture. Users suffer from inconsistent experience. From one interaction to the next, users are exposed to site-specific experiences that are distinctively different, like being taken around the world from click to click. *Thirdly*, task template is a rigid metaphor. There is only one 'condition' and one 'action' setup in the template. The lack of support for multiple conditions and multiple actions in sequence may be too restrictive for broad adoption. For example, the current task template cannot handle the following use case: everyday at 6:00 a.m. [condition 1], if IBM stock price is > $180 [condition 2] and US currency exchange rate is < 89 cents [condition 3], then post on Facebook with a message "stock doing good" [action 1], followed by a tweet [action 2], followed by notify colleagues [action 3]. *Fourthly*, today's tasking platforms are positioned as Apps-integration platforms. They are not designed with the web in mind. Their approach is not open, making the scope very restricted.

5 Task as a Service

This section discusses *web tasking over the cloud* and illustrates how this approach does not have the challenges identified in the previous section. It also proposes adding a Task as a Service layer to current cloud platform.

5.1 An Open Tasking Model

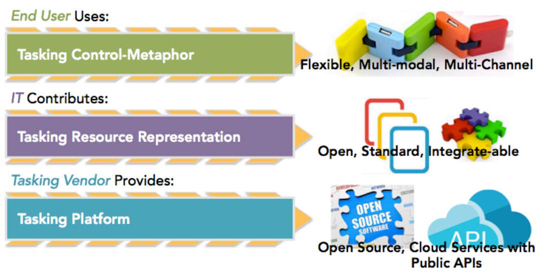

End User Uses:
Tasking Control-Metaphor
Flexible, Multi-modal, Multi-Channel
IT Contributes:
Tasking Resource Representation
Open, Standard, Integrate-able
Tasking Vendor Provides:
Tasking Platform
OPEN SOURCE SOFTWARE
API
Open Source, Cloud Services with Public APIs

Figure 8: An Open Tasking Conceptual Model

For tasking to be as widely adopted as web browsing, the tasking conceptual model must be open. An open tasking conceptual model has the following characteristics:

- The tasking platform must be ready for open source with well-defined APIs. This enables third parties to build alternate Tasking Control Metaphor on top.
- The Tasking Resource Representation must be open and standard-ready. The representation itself must be a declarative artifact with a meta-model. It should not be a programmatic artifact. A declarative artifact, like HTML, provides a method for third party tasking platforms to inter-change.
- The tasking control metaphor must be flexible, able to support any device of interactions and is open to support multiple metaphors and multi-modal of expressions.

5.2 Web Tasking: An Open Tasking Model for the Web

Web tasking [7,8] is the notion of extending the current web architecture for enabling end users to freely task with web resources from across the web, as freely as they browse, as in the web-browsing paradigm. Web tasking distinguishes itself from current tasking approaches like IFTTT or Zapier etc. in the following ways. *Firstly*, its architecture is in full compliance of Fielding's web architectural principles [13], therefore is designed with a web scope in mind to interoperate generically across the web. Its architecture is also open and without domain specificity. For example, any web tasking resources are URL addressable web artifacts, which is not the case for Recipe of IFTTT or ZAP of Zapier. *Secondly*, because tasking resource representations are web artifacts, they can be linked together by hypermedia links. For example, actions can be chained together in simple hypermedia links. Therefore it is able to support a tasking control metaphor that is as flexible as hypermedia links actions.

5.3 BOTbit: A Universal Model for Tasking Resource Representations for the Web

Web tasking architecture has a meta-model called REAST [8] that is representational action-state transfer. It is designed to represent actions of resources instead of the data-instance of resources. BOTbit's meta-model is an added media-type. This enables open and standard-ready prescribed steps for developers to contribute web-tasking resources with built-in interoperability. REAST's standardized Action interfaces, and its attribute-model for web resources, enabled simple widgets to be built as units of control to express actions and conditions that average users can handle. It also enables a flexible control-metaphor such as jigsaw or train to be built for average users.

Just like HTML has its universal representation meta-model in the web-browsing paradigm, and it is pivotal for its web-scope universal operations, in the same token, in the paradigm of web tasking, it has a Tasking Resource Representation that is universal across the web, and is pivotal for web tasking. *Firstly*, it enables different vendors Tasking Platform to process each other's Tasking Resource Representation instances. *Secondly*, it provides an abstraction needed such that the web tasking platform can built a universal tasking user experience and sparing the users from the site-specific content from one interaction to another.
I coin this universal model for Tasking Resource Representation in our Web Tasking implementation "BOTbit" (that is a bit of resource for the web roBOT). BOTbit in web tasking is analogous to HTML in web browsing.

BOTbit as Web Tasking's Tasking Resource Representation model has the following major parts:
(1) It has a universal ***action representation*** of resources with ***standardized action interface***, namely, create, read, update, delete and others. This part is called REAST (REpresentational Action State Transfer) [8]. Imagine turning "Weather Resource" into a web tasking resource, this Weather BOTbit instance will have one action representation, "Read". User can task by adding a Schedule-condition to create a task with the semantic of "<condition> everyday at 6:00 a.m., <action>read weather".
(2) It has a ***resource attribute collection*** that is mapped back to original data sources. This portion is designed for the automatic generation of user-forms used in user-author conditional expressions. Using the weather BOTbit again, one can say "<condition> When weather.status = "rainy", <action> notify friends".
(3) Other elements like BOTbit ***graphical Icon***, mnemonic labels, and others.

This universal web tasking resource model enables interoperability of resources of diversified types and sources, enables cross domain interactions and is an inter-changeable web artifact to be processed by machine (for web automation) or by human [12].

5.4 Scribble: A Multi-Modal Tasking Control Metaphor

Web tasking [7] uses a tasking control-metaphor called "Scribble". I call this a multi-modal tasking control-metaphor because users can scribble by text (the "natural language" control-metaphor), or scribble by widget using the jigsaw puzzle control-metaphor.

Whether scribble by text or by widgets, there are three units of controls supported by Web Tasking Platform: they are namely, "*Action*", "*Condition*" and "*Schedule*". Sequence of actions are linked together by hypermedia links between. For each "Action", there can be zero or more "Condition" widget attached. "Schedule" widget has to be placed next to the first "Action" widget, which control the schedule for the execution of the entire action sequence. User selects the widget type, snap in a user-chosen Web Tasking resource icon to complete the scribble ready for submission to the Tasking Platform.

The tool that provides end users scribble support is called the *Web Tasker*, analogous to the function of Web Browsing in the browsing paradigm.

5.5 An Open Architecture for Tasking Infrastructure

The middleware of the Web Tasking Platform is called ***Web Interaction Server*** [14]. Analogous to the web application server for web applications, Web Interaction server is a server side engine, built to support BOTbit as the web tasking resource model. It has open APIs and is open-source-able and standardize-able.

Because of the open architecture of Web Interaction Server, existing Apps can call these Web Interaction Server APIs and add Web Tasking capabilities to enhance existing Apps as a hybrid. This is what IFTTT and Zapier cannot do without a web compliant architecture.

5.6 TaaS: a Cloud Layer for Users

Current cloud layers are namely, Infrastructure as a Service (IaaS), providing storage, computing capacity and connectivity as services, Platform as a Service (PaaS) providing cloud operating developing environment as services, and Software as a Service (SaaS), providing application solution services, such as healthcare solution, commerce solution etc [1]. This is a highly efficient development environment for developers who can compose applications by assembling existing services to create Apps for their users.

This paper proposes to add Task as a Service for end users to access web tasking resources to compose their tasks. By using a web-tasking resource admin tool, current APIs from the API economy of the SaaS layer can be converted into BOTbits, transforming them into web tasking resources for end users to assemble tasks with.

The TaaS layer can also be a seamless integration layer to interoperate with other vendors' cloud platforms. For example, APIs from the SaaS layer of another cloud vendor can be transformed into corresponding BOTbits for the local TaaS layer, made available to its end users the same way without end users noticing that these BOTbits are from foreign cloud environment, making cross vendor cloud interoperability seamless.

TaaS layer conceptually should contain the following logical components:

- ***A Web Tasker***: a tasking environment for users to do web tasking (aka to scribble). TaaS needs to add billing and metering to the web tasking.
- ***A Web Tasking Community***: We have a tasking community to share scribbles with others and to consume scribble produced by others
- ***A BOTbit Repository***: to collect a list of local (to the SaaS layer) and remote (from other vendor's SaaS layer) web tasking resources that TaaS users can access
- A third party Governance Infrastructure of BOTbits needs to be put in place to ensure the trustworthiness of services across the cloud.

6 Conclusions and Future Work

In this paper, I argued that because programming is still central to the control-metaphor of end user programming approach, visual programming and the mashup approach, a drastically different software engineering approach is needed to enable accessible controls for average users for their own tasking using cloud resources.

In this paper, I named the recent trend from IFTTT and Zapier as the "Tasking" approach and provides a Tasking Conceptual Model, asserting that because the control-metaphor in the Tasking approach is from the end users' domain, it is more accessible to end users. After pointing out the challenges of these popular tasking approaches, this paper also proposed using web tasking on the cloud as an alternative, pointing out the architectural advantage of web tasking being compliant to the web principles, therefore reaps the benefits of openness, and interoperability. This paper also called out the pivotal importance of having a universal model for Tasking Resource Representation in order for web wide operations without domain specificity, even including the Internet of Things. This paper also took this further to propose adding TaaS as a cloud layer for users.

Tasking and TaaS is just at its infancy. However, the pressing user requirement to demand more accessible controls over IT that is consumable for end users has been validated by the overwhelming user responses to these early implementations such as IFTTT and Zapier. This paper calls out the need for Tasking and TaaS as a research area that warrant focus and top attention.

There are challenges that should be considered for future study. Self-efficacy validation, end user testing, debugging, trust, privacy, security are key areas that warrant much attention ready for enterprise consumption.

References

[1] M. Litoiu, M. Woodside, J. Wong, J. Ng, G. Iszlai. A Business Driven Cloud Optimization Architecture. *Proceedings of the 2010 AGM Symposium on Applied Computing*, pp. 380-385, 2010.

[2] A.J. Ko et al.. The State of the Art in End-User Software Engineering. *ACM Computing Surveys (CSUR) Volume 43 Issue 3, Article 21.* April, 2011.

[3] IFTTT / Put the internet to work for you : https://ifttt.com/.

[4] Automate the Web – Zapier : https://zapier.com/.

[5] Node-RED : http://nodered.org/.

[6] M. Blackstock, R. Lea. Toward a Distributed Data Flow Platform for the Web of Things (Distributed Node-Red). *Proceedings of the 5th International Workshop of Web of Things*, pp. 34-39, 2014.

[7] J. W. Ng, D. H. Lau. Going Beyond Web Browsing to Web Tasking: Transforming Web Users from Web Operators to Web Supervisors. *Proceedings of IEEE ICWS Workshop on Personalized Web Tasking (PWT 2013)*, pp. 126–130, 2013.

[8] J. W. Ng. Adapting REST to REAST: *Building Smart Interactions for Personal Web Tasking. Proceedings of IEEE Services 2014, 2nd International Workshop on Personalized Web Tasking (PWT 2014)*, pp. 38-46, July 2014.

[9] A.J. Ko, B.A. Myers, H.H. Aung. Six Learning Barriers in End-User Programming Systems. *IEEE Symposium on Visual Languages and Human Centric Computing*. pp.199-206, 2004.

[10] J. Wong, J. I. Hong. Making Mashups with Marmite: towards end-user programming for the web. *Proceeding of the SIGCHI Conference on Human Factors in Computing Systems*, pp. 1435-1444, 2007.

[11] Scratch: https://scratch.mit.edu/ .

[12] R. Verborgh, E. Mannens, R.V. de Walle. The Rise of the Web for Agent. *Proceedings of The First International Conference on Building and Exploring Web Based Environments (WEB 2013),* pp. 69–74, 2013.

[13] R. N. Taylor. Principled Design of the Modern Web Architecture. *ACM Transactions on Internet Technology (TOIT)* 2(2):115–150, 2002.

[14] J. Ng, D. Lau, T. Ng. Web Interaction Server: A Web Tasking Middleware: A Position Paper. Proceeding of 24th Annual International Conference on Computer Science and Software Engineering (CASCON 2014), pp.303-305, 2014.