How will we interact with the #WebWeWant?
Position statement for the 5th International USEWOD Workshop: Using the Web in the Age of Data, May 31st, 2015, Portoroz, Slovenia.

Position paper

Fostering intelligence by enabling it

Ruben Verborgh
Postdoctoral researcher at Ghent University – iMinds

In a couple of months, 15 years will have passed since Tim Berners-Lee, Jim Hendler, and Ora Lassila wrote the Scientific American article "The Semantic Web". It's hard to imagine that, another 15 years before this, the *Web* didn't even exist. The article talks heavily about *agents*, which would use the Web to do things for people. Somehow, somewhere, something went terribly wrong: the same time needed for the Web to liberate the world has hardly been sufficient for the Semantic Web to reach any adoption. And still, there are no agents, nor are there any signs that we will see them in the near future. Where should we even start?

Even if the visionary Semantic Web article was just a guiding example, we still seem to be quite far from the envisioned intelligent agents. Fortunately, we're progressing. The Semantic Web has brought us many useful things already:

- lots of **Linked Data**—but we mostly need to download datasets locally if we want to do something useful
- **intelligent servers** such as SPARQL endpoints—but most of them suffer from high unavailability rates
- lots of **research**—it will be practically applicable if only data and services become available when we need them

While there are some disturbing "if"s and "but"s, the above brings hope: apparently, **the necessary building blocks are already there**. So where are the agents? I'm not the first to ask: Jim Hendler, co-author of the original vision, wonders the same.

While the technological stack might be there, there are still several obstacles to overcome, some of which we have created (and continue to create) ourselves. In particular, our services are **too heterogeneous**, are **not self-describing**, and try to be **too intelligent**. Let's look in more depth at those issues—and how we might solve them.

Heterogeneity confuses clients

Approximately half of the talks at API conferences start with a variation of this slide: *"we're doing great, because there are more than 12,000 APIs"*. Personally, I never understood how **overgrowth** can be a *good* thing: 12,000 APIs means 12,000 *different* ways of engaging in machine-to-machine interaction.

Imagine if every pen required a different writing style. If every car came with its own set of pedals. If every tap had a new way of getting water. We would never write, drive, or drink—or actually, we might just be able to cope, since **humans deal with change well.** And while different APIs indeed have different domains, does that warrant an entirely new interface? Clearly, an ice cream van and a tractor have totally **different domains and purposes**, but yet their **interface is largely the same**. Once you know one, the other becomes fairly easy. The same cannot be said about Web APIs.

A major problem with machines is that they don't deal well with interface heterogeneity. They like it when things are **rigidly structured**. That's why we have a strict data format like RDF: all facts are expressed as triples. If something cannot be represented in a triple, you need to break it down into parts that can be. And this works for data from any domain—so why can't it work for APIs from any domain?

Instead of treating APIs like monolithic, give-or-take marble rocks, we should think of **reusable building blocks** to compose APIs. An ice cream van and a tractor both have a gas and brake pedal, a horn, a steering wheel, wipers, flashers, and more. It makes total sense to reuse them. The few features that are different can be controlled separately. If we apply the same way of thinking to APIs, the notion of *generic* clients, or intelligent agents, becomes much more realistic.

Could you use an interface without controls?

Back to the ice cream van. Would you be able to drive it if the pedals and steering wheel were **invisible and untouchable**? As in: they are physically there, you can manipulate them, but you cannot see nor feel them. How would you position your hands and feet?

This is how agents feel when they try to consume **non-hypermedia APIs**. No clues as to where they are or where to go. It's like a web page from which all links have been removed. Once you're on a page, it's impossible to leave—unless you have studied the manual. But honestly, when was the last time you needed to read a manual before being able to use a website?

Recall that machines have more trouble **dealing with unfamiliar environments** than humans. So if one party certainly needs beacons for navigation, it's the machines. We humans could still find a way around: for instance, if this page did not contain any links, you could manually try to edit the URL to go elsewhere. That's why I find it puzzling that Web APIs for machines usually have far *less* hypermedia controls than websites for humans, even though machines need them the most. Perhaps that's because API creators assume that humans will hardwire clients to consume them… but that does not bring us any closer to generic clients.

Simple servers, clever clients

Some API developers are well aware that the current generation of clients is limited. They try to **compensate for clients' limitations** by building servers that are as intelligent as possible. The service does all the work, the client just has to perform simple calls.

While this seems nice, there are possible problems. First of all, the server doesn't know **what the client wants to do**. Maybe you want to race with the ice cream van, or sell chocolate milk instead. It's technically possible, but does the interface allow it? At a high level, what all APIs do is providing access to some kind of database (for a very liberal

definition of "database"). The more the API tries to do, the less *clients* can do with the data, because everything has already been decided. Second, this can also have dire consequences for the server: much work means a high per-request processing cost. Uniform data models like RDF work very well, but the uniform query language SPARQL comes with serious availability problems—at least partly because it is so expressive.

Clever clients do not require an intelligent server, they require a server that *enables* them to act intelligently. We need to think in a different way about APIs for that: **reusable building blocks that explain themselves** are an important starting point. Instead of being preprogrammed for a specific task, clients could ask a server *"what blocks do you have?"* and *"how can I use them?"* Clever clients should decompose a task into elementary subtasks and use the building blocks they know to solve them.

And that's, in my opinion, a **key gateway to intelligent behavior**. Like most of you, I've never operated an ice cream van, but I'm pretty sure I could do so should I ever get the opportunity. That works because it offers me the building blocks and the interface, and I possess the ability to piece things together. Clearly, intelligence needs to be enabled.