Position statement for Dagtuhl Seminar 23081 on “Agents on the Web”

Julian Padget

The research questions posed by the organizers are real and underline the significant task that are faced in bringing about the vision of agents on the web. It is notable that each of the RQs addresses a different aspect of design, with the collective goal of “getting things done”, including how collective behaviour may be governed. The element of design that is perhaps implied, but not explicitly addressed is when getting things done does not work. Of course, there is reasoning (RQ3) and planning, which of itself implies the possibility of plan failure and replanning, but such elements still depend upon everything else continuing to work at some level. And RQ4 cites policies and norms, social constructs and the monitoring and regulation of agents and people, which also largely depend on other things working.

- What good is “intelligence” if it can only cope with “known knowns”?

The problem (among many!) that interests me is how do we engineer for varying degrees of partial or complete (sub)system failure. This is not just a matter of testing, since as we all know, test coverage will never be enough and as Dijkstra opined “testing proves the presence of bugs, not their absence”. Neither is engineering for resilience enough (known unknowns): it is necessary, but not sufficient.

In building agents on the web, the fundamental tenet is decentralization. What one agent cannot do, because it cannot do everything, another one, or some service, if it can be found, will do it on behalf of the agent: we hope for a form of society of agents and services. This in turn creates networks of dependencies, driven by a variety of intentions: elegance (we are engineers!), efficiency, avoidance of unnecessary duplication. But dependence, or the lack of independence, shifts points of failure. Although that still leaves in the realm of classical safety engineering.

- What happens when the data stops?
- What happens when the service stops?

The more subtle problems, as Lamport taught us, are not when something stops, but when something functions outside of specification:

- What happens when the data is wrong?
- What happens when the service does not return a correct result?

Local solutions within an agent or a service can deal with some of the above, but these questions restrict our focus to what happens inside components, using conventional software engineering approaches. The unknown unknowns that the system (of systems) needs some capacity – not capability, or they would be known – to handle are a function of complexity, whose distinguishing feature is emergence, resulting from what happens between, rather than within software components. This can push the system (of systems) into self-reinforcing good or bad states.

- How can such trajectories be detected and acted upon?
- How to govern governance?
- How can systems (of systems) mitigate unknown unknowns?
- How can systems (of systems) recover from unknown unknowns?
- How is human responsbility and accountability maintained is such systems (of systems)?