Introduction

The imagination of machines as human-like and of humans as machine-like has been a central facet of modern Western technical culture, most prominent in the kind of “thinking” computers do and the kind of “computing” that human brains do, but also spanning analogies between capacities for decisions, rationality and control or self-regulation. Arguing that metaphors, analogies and images, far from being mere poetic decoration, run deep in reasoning and in how the world thus becomes organised, one of the animating agendas of critical studies of technology has been a search for “a deeper, broader, and more open scientific literacy” (Haraway 1997: 11), for a more metaphorically aware “critical technical practice” (Agre 1997), and for spaces to intervene in the cultural imaginaries of technoscience (Suchman 2007: 227).
The well-worn opposition of the “real” versus the “merely metaphorical” beckons, however. As does that of “figural” and the “literal.” For Agre, Haraway and Suchman, these oppositions did important work in challenging realisms, opening out technical culture to critique, and in Haraway’s case, examining the terms in which Western culture imagines the real in the first place. In this chapter, however, I want to explore the value of another contrast. If we understand “figurations” in Haraway’s words, as “performative images that can be inhabited” (Haraway 1997: 11), we might contrast figurations with instrumental metaphors, the latter less a matter of habitation, and more one of associations available for use.

Writing from a field almost entirely disconnected with that of Haraway, Daniel Dennett argued that the apprehension of something (be it person, animal or machine) as an agent with intentions and reasons of its own is itself a finely tuned and adaptive trick, this “intentional stance” an acquired technique for living a life in common with others (Dennett 1989). This provides one reading of what Haraway refers to as the “tropic quality of all material-semiotic processes” (Haraway 1997: 11). Seeing others is always a “seeing as,” then, and more direct than an intellectual equivalence; in Kukla’s reading at least, a stance is to be understood practically, as “a way of readying your body for action and worldly engagement” (Kukla 2017: 4). Figurations understood as stances rather than associations, would be just such “ways of readying,” constitutive of relations to others.

Dennett also described a “design stance,” sitting somewhere between the intentional and the physical, a mode of interpretation that sees something as embodying a purpose, a normativity: that which it is there for, an embodied intention, its history coupled up to its behaviour according to what it is supposed to do (Millikan 2000). Crucially, for Dennett, these stances are themselves designed, indices however not of a master creator, but of the long and blind process of evolution. The result: the installation of pragmatic instincts that enable animals like us to both cope in and create a world of social and technical complexity. Such entanglement of figurations in species history further distinguishes them from the metaphors we use. The emergence of tool use and complex sociality are part of the same story as the emergence of the forms of worldly engagement according to which some beings are approached as intentional or purposeful, setting others in relief as things.
My interest in this chapter concerns the circumstances in which what counts as an intentional agent are reconfigured, the ongoing and unfinished history of technology-human entanglement. I look to a domain largely invisible to critical studies of technology: that of configuration management. In the 1990s, growing complexity of computing environments led to the relations between configuration managers and the systems under their stewardship being called into question. I examine “Promise Theory,” a philosophy emerging from configuration management which treats computers as intentional agents. Promise Theory has been ignored by the social sciences, but its circumstances of origin are very familiar: the heterogeneous infrastructures of scientific computing of the early 1990s, circumstances that also inspired Susan Leigh Star’s widely influential relational approach to infrastructure. The common origins are revealing, as are common concerns with locality and distributedness. In reformulating relations with machines, metaphors of course abound, of puppets, engines, immune systems, orchestras. But we can also, I suggest, detect figurational shifts, re(con)figurations, shifts in forms of worldly engagement such that “things making promises” is more than a manner of speaking, a form of stewardship of distributed systems.

**Figuring Configuration Management**

Configuration management originated as a set of techniques developed in support of systems engineering, pioneered in the 1950s by NASA, in order to keep track, make manageable and make auditable the vast swathes of stipulations that accompany complex technical systems, asserting how they should be set up, the states their component parts ought to be in, dependencies that should be present, the versions that ought to be used, settings that should be set, switches flicked and plugs plugged (Watts 2011: 10). A vast exercise in paperwork (nowadays usually virtualised in configuration management databases) which is essential to the smooth running of innumerable infrastructures and platforms, yet which almost never surfaces into wider awareness. An infrastructure’s infrastructure, if there were such a thing.

In the 1990s, configuration management in information technology made a subtle but significant departure from this broader tradition,
associated with the transformative effects of automation. A new kind of configuration management tool appeared for managing the configuration of networked computer systems, which would systematically check whether systems under its stewardship conform to “policy” (the officially recorded configuration) and take remedial action to fix discrepancies. With automated apparatuses serving as their eyes and hands, systems administrators became the designers and operators of sophisticated automation infrastructures.

The first widely used automated configuration management tool was CFEngine. It was developed in the early 1990s by Mark Burgess, a theoretical physicist working at the University of Oslo. He released the software open source in 1993, and by the late 1990s it had become by far the most widely used tool of its kind. Over the next two decades CFEngine went on to serve as archetype for a class of tools that would redefine the nature of configuration management in information technology. These would later include Puppet (released in 2005), Chef (released in 2009) and Ansible (released in 2012). The early success of CFEngine drew Burgess away from physics and into the world of systems administration, and he continued to develop the software alongside a theory of configuration management in the years that followed.

As a postdoctoral scientist, one of Burgess’ duties had been the administration of his research group’s network of computers. This was a classic situation of, to use the unwieldy catchphrase of the time, heterogeneous distributed computing: the kinds of machine used in research had a wide range of configurations, versions of software, permissions associated with user accounts, file system structures, scheduling of batch processing, and so on, often different models of device, typically running variants of UNIX.

Facing the challenge of managing such heterogeneity, Burgess looked to apply automation to the tasks of configuration management. The standard approach of the time involved curating custom procedural scripts. The activities a system administrator would otherwise carry out manually would be written as step-by-step algorithms that could be executed from a privileged centre of control. This “imperative approach to thinking” (Burgess 2015b: 2) turned out to be fragile. The diversity of computing environments made these scripts complex in their own right, unwieldy to maintain and “brittle,” tending to produce unpredictable results,
especially when run against a machine in an unknown state (Spencer 2015). Luke Kanies, who would later author Puppet, dubbed the challenge of maintaining assumptions a problem of “software rot” (Kanies 2003: 119). Burgess, on the other hand, interpreted the problem as a physical one, the idea that in an unpredictable world, commanded systems will tend to diverge from a known starting point (Burgess 2015b: 4).

So CFEngine moved away from procedural scripting. It was based instead on a declarative approach. The desired configuration is stated in a special syntax, without stipulating what to do, how to check, enforce, or make a repair in relation to it. A set of CFEngine’s specialised “software agents” would then interpret and compare this “policy” against the observed state of various computers and generate contextually specific steps for remedial action if necessary. The activities of these autonomous agents were not intended to fix problems as a one-off complete repair. Rather, they were intended to run in the background in a decentralised fashion, producing over time a \textit{convergence} between the actual and the proper state of affairs.

This combination of declarative policy, decentralised automation and convergent repair became the paradigm for IT configuration management and with huge impact beyond. The automation of configuration is, for instance, the heart of cloud computing. Over the last two and a half decades, the paradigm expanded to provide comprehensive infrastructure automation, so much so that configuration management tooling exceeded its original purpose. Beyond just checking and repairing systems that already existed, tools like CFEngine also provide the means to spin up new infrastructure on demand, declaring it into existence, as it were. “With CFEngine,” an Automation Engineer at LinkedIn is quoted as saying, “I can define a new Software Defined Datacenter and offer IAAS [infrastructure as a service] and PAAS [platform as a service] to my customers within 10 minutes” (CFEngine n.d.: 3). The infrastructure of infrastructure indeed!

The automation pioneered by the configuration managers paved the way for wider suites of tooling, which brought automation to code management, release pipelines, build processes, testing cycles and deployment. Together these tools became the technical foundation for agile and continuous delivery-based methodologies, which in turn transformed the manner in which functionality is delivered in digital environments,
taking us from the “old approach” of delivering new versions of applications after long periods of stasis, to the current paradigm involving constant flows of small iterative changes (Humble and Farley’s textbook Continuous Delivery being the enduring reference point for the new technical foundation; 2010). When Neff and Stark described the agile approach as “permanently beta” back in 2004, they saw it as an approach to design (2004). With the rise of automation and cloud infrastructure in the years that followed, however, we can now say it has expanded significantly in reach, an approach to the whole delivery lifecycle: not just design but also delivery and the operation and management of the systems that result.

It is testimony to the depth at which configuration management is embedded that it has escaped attention of critical studies of technology even in cases where they address cloud computing square on. Peters, for example, takes a user-centred view, associating cloud computing with cloud storage (Peters 2015). Hu and Bratton go in the opposite direction, with their focus on the evolution of physical infrastructures, a broad story of communication networks and datacentres, without addressing the question, missing in the middle, of the techniques by which it has become possible to tame that complexity (Hu 2015; Bratton 2016). Amoore similarly examines analytics and algorithms in light of material infrastructure, but leaves little space for understanding the “how” that makes these techniques possible (Amoore 2020). None of these thinkers appreciates the problematic of machinic autonomy that underlay the ability to craft self-regulating, automatically provisioning systems for computational infrastructures, and which is now intimately woven with digital culture.

Smart Intentional Infrastructure

Automating configuration management was not simply a matter of finding clever ways to script manual tasks. It required and fostered reinterpretation of the problem of configuration itself, which became a topic of debate and discussion among an emerging community of IT configuration managers, on mailing lists and at conferences. For Burgess, this intellectual project led towards the development of a theory of cooperation he
called “Promise Theory.” Promise Theory arose out of Burgess’ attempts to formulate what it was he had been trying to do with CFEngine and became over the years a lot more than a theory of configuration.

Moving from the context of theoretical physics into the professional community of system administration, Burgess reports that he encountered a set of intuitions about computers that seemed to be aligned with the procedural scripting approach to configuration management. This idea, that computers were like obedient rule followers, rubbed awkwardly against his more physical, more cybernetically inflected intuitions. Writing about his experience at the conference in Large Installation System Administration (LISA) in 1997, Burgess relates that

To me, the work I presented was just a small detail in a larger and more exciting discussion to make computer systems self-governing, as if they were as ordinary a part of our infrastructure as the self-regulating ventilation systems. The trouble was, no one was having this discussion … In the world of computers, people still believed that you simply tell computers what to do, and, because they are just machines, they must obey. (Burgess 2015a: 4)

Though first by its name an “engine,” when he returned for the following year’s LISA, Burgess had armed himself with a new metaphor to cut through these preconceptions. His talk, published as “Computer Immunology,” went on to be influential in the field. It used the metaphor of an organism and its immune system to perturb system administration thinking away from its familiar notion of the obedient computer. “CFEngine,” he wrote,

fulfills two roles in the scheme of automation. On the one hand it is an immediate tool for building expert systems to deal with large scale configuration, steered and controlled by humans. It simplifies a very immediate problem, namely how to fix the configuration of large numbers of systems on a heterogeneous network with an arbitrary amount of variety in the configuration. On the other hand, cfengine is also a significant component in the proposed immunity scheme. It is a phagocyte which can perform garbage collection; it is a drone which can repair damage and build systematic structures. (1998: 287).
The proper behaviour of cells which sustains the life of an organism is not enforced by their strict obedience to commands. While almost all cells do have a catalogue of instructions, some may have faulty DNA, something might go wrong in the reading, or a foreign agent may be interfering with the normal processes of interpretation. The immune system is comprised of mechanisms capable of detecting and responding to aberrant behaviour, keeping things healthy at a higher level of organisation. Similarly, CFEngine is built on scepticism that commands can be sufficient to ensure convergence: machines may have missed instructions, might end up with multiple conflicting instructions or might be missing dependencies for carrying them out. Instead, it implemented a set of autonomous agents capable of identifying problems, making remedial changes or bringing issues to the attention of the administrator.

Later reflecting on this period of time, Burgess notes that he soon abandoned the immune metaphor in favour of a theory of promises based on the figuration of computers as intentional agents.

The idea gelled in April of 2004 that autonomously specified declarations of intent were simply promises—something conceptually opposite to obligations, or any other kind of declarative or imperative logic. Promises could be defined as a network that was not necessarily the physical network between computers, more like a network of self-imposed constraints that we call intentions … Emerging was a theoretical model for a kind of smart, intentional infrastructure based on graphs of autonomously made promises. This graph theoretical approach was an altogether more plausible and scalable approach to locality than deontic logic. (Burgess 2015a: 247)

Burgess maintained the contrast with imperative thinking within the theory of promises: “obligations” can be understood as a special case, as promises made for others, fragile in comparison with promises one makes for one’s self. This lexicon was implemented in CFEngine’s third version: policy was to be understood and encoded in terms of the promises that machines or systems make (to one another, and to users). And in addition to developing CFEngine, Burgess also built out this theory, developing it into a graph theoretical framework for the analysis and design of intentional relationships in distributed systems, in collaboration with Jan Bergstra, a Dutch computer scientist.
To translate into more familiar sociological terms, promises formalise the normativities of technical systems: what it is that they are supposed to do. But instead of using the design stance, Promise Theory construes technical normativities via the intentional stance. They are treated as a thing’s intentions rather than purposes that can be read into it. This shift between stances is strategic: the problem with addressing technical normativity via the design stance is that it is too easy to regard the purposes of designed things as residing in the mind of their creator, or else in some separate source of authority, such as the configuration management database. By locating the source of normativity outside of the technical thing, the problem of configuration management seems to be, indeed, one of imposing the correct behaviour on subservient infrastructure via obligations. The language Burgess and Bergstra deploy, however, in formulating things’ “autonomously made promises” affords things a depth of their own, enabling us to see their purposes as their own, as local to them. This localism is central. Promise Theory, they write, “is a relativistic theory of ‘many worlds’ belonging to its many agents” (Bergstra and Burgess 2019: 4). Where small and simple networks of computers could be controlled with impositions, the large-scale complex networks of contemporary information technology ought to be treated, in design and in maintenance, as “smart intentional infrastructure.”

The shift to the intentional stance of course raises questions of whether things “really” have intentions. Is this just a “figure of speech”?

Perhaps this makes you think of promising something to a friend, but don’t forget to think about the more mundane promises we take for granted: a coffee shop serves coffee (not bleach or poison). The post office will deliver your packages. The floor beneath you will support you, even on the twenty-third level. If you could not rely on these things, life would be very hard. (Burgess 2015b: 39)

We are to think of promises, then, as environmental as well as explicitly designed (coffee is not designed as such), as embedded into surroundings by the particularities of cultural and technical histories. The language of Gibson’s ecological psychology would not be out of place here, for promises are relational to the kinds of bodies and purposes that come into articulation with them, and that may foster them over time (in the
terminology of the theory, promises are made to specific promisees, not to the world in general). The post office’s promise to deliver is addressed to some kinds of beings in its vicinity; for others, its promises may be quite different (its guttering promising perhaps a place to roost).

In certain places, however, the concept can seem rather thin: “an intention is nothing more than the selection of a possible outcome from a number of alternatives, based on an optimization of some criterion for success” (Bergstra and Burgess 2019: 7). They note that promises are intended to capture the sense in which inanimate objects “serve as proxies for human intent” (Bergstra and Burgess 2019: 14; also Burgess 2015b: 9). Are they in danger here of falling back into imposed intent, against their own notion of locality? If a promisor is a proxy, then we might indeed question whether there it has much real autonomy.

Bergstra and Burgess do fall back upon a “default” apportionment of intention as naturally human. We might, however, read this as a sign of context, an anticipation of aspersions of animism, the interpretive effects of being seen to confuse categories that have held fast in the West for millennia. There is of course a vigorous literature on the cross-cultural nature of apportionments of agency and perspectives between humans and non-humans (notably, Viveiros de Castro 2012; Descola 2013): not, however, a literature with which their readership is likely to be familiar.

Instead of interpreting this being a proxy as implying that intent is derived from some particular humans that can be located and pointed to, we might more generously interpret it as referring to the embedding of technical things in the world, the historical sedimentation of selections that forms worlds. It is not as a physical object that a technical thing has its intent, but rather as an historical object. In cases where a designer can be pointed to, there is of course a particular human agent involved in this history, but even here, many elements combine which have unwritten and tacit pasts of imitation, inspiration and copying, as well as the historicity of cognitive capacities involved in the designing (Millikan 1984). A humble webserver would then bring together many threads of intent: that of its designer, the architect of its implementation, but also the history of imagining computers as network endpoints, the concept of service and the history of servitude.

The appeal to a depth of intent in formulating the problem of configuration management is not, I suggest, a simple rhetorical move, making
use of metaphor as a route to a better explanation. It draws on the deep history of relating to other beings as having intentions and purposes, the history of the figurations we inhabit, of sharing space and co-habiting a world alongside others with intentions of their own. It is significant, I would suggest, that Promise Theory emerged from the relationship of configuration managers with the increasingly complex infrastructures under their care, rather than as an intellectual project driven by an abstract problem. Where configuration management had previously entailed a relationship of imposed order, the complex distributed systems of the 1990s (and since) entailed a subtle shift, in which those systems’ contingencies, the fact that they may always be in a state other than the proper one, came to have new significance: no longer an invitation for a corrective intervention, they required stewardship and care (see also Kocksch et al. 2018). The historicity of relations with technology is thus doubly entwined with the historicity of technical function (Spencer 2021): firstly in the figurations of technical systems as purposeful and intentional, and secondly in the novel possibilities these shifts open up.

Like CFEngine, Promise Theory participated in the development of infrastructures more widely, most obviously where it is directly cited as inspiration or support. The approach, for instance, is named as the basis for managing policy in the networking giant Cisco’s “intelligent networks” (Cisco 2014). A second example comes from nearer to home. Adam Jacob, who had originally developed the “Chef” configuration management system, described his thought process in devising plans for a new application automation system he named “Habitat”: “I think there is an application problem. I think we are thinking wrong about the shape of the application. And what if … applications could behave as well-behaved actors in like a promise theory sense? And what would be the promises that those applications make to each other? And from there it led to Habitat” (Jacob 2018, np). Treating applications as actors rather than as software (which begs the question of whether they are running properly), Habitat provides an infrastructure for the mutual monitoring of applications and their ability to propagate information about each other through “gossip.” The intents embodied in these infrastructures have many sources, but it is not a stretch to include among their number a “promise theoretical” refiguring of machinic intent.
Figuring Infrastructure

While Promise Theory (and configuration management in general) may have escaped attention among critical scholars of technology, the circumstances in which automated configuration management appeared are rather familiar. In addition to being the site of emergence of CFEngine, scientific computing environments of the late 1980s and early 1990s had a formative influence on the development of social approaches to the analysis of infrastructure.

It is a truism of science studies that scientific practice is not a singular phenomenon, but consists of a plurality of epistemic cultures (for instance, Galison 1996; Knorr-Cetina 1999). On a more mundane level, as Burgess puts it, in the sciences “every kid is special”: because research agendas can point in their own unique directions, demands for specialist equipment and unique configurations can readily override institutional pressures to adopt standardised technology. With tendencies towards computational heterogeneity built in, it is no surprise that CFEngine and modern IT configuration management emerged from a university research context rather than, for instance, the IT departments of commercial organisations, or Silicon Valley.

Another infrastructure very much of this moment was the Worm Community System (WCS). WCS was an information sharing tool, funded in the US by the National Science Foundation, and designed for the global research community studying C. Elegans, a nematode worm widely used as a model organism by molecular biologists. On the WCS team were two ethnographers, Susan Leigh Star and Karen Ruhleder, whose participation in and analysis of the process of implementation proved profoundly influential in the development of sociocultural research into infrastructure, maintenance, computer-supported cooperative work, standardisation and communication (Star and Ruhleder 1996). What they called, with the gerund-ified flourish of grounded theory, “infrastructurin,” named that situated and practical process of becoming infrastructure which was both goal and problem for WCS.

Making sense of the WCS project, Star and Ruhleder argued, required a relational approach to infrastructure. In a departure from emphases on wide arcs of technological development, and the stabilisation of designs
(e.g. Bijker et al. 1987), Star and Ruhleder foregrounded the embeddedness of infrastructure, its intrinsic relationality with and in sites of practice. Technologies become infrastructures as a contextual achievement, in which their use becomes integrated into routines of practical activity, so much that they are no longer explicitly put to use, becoming “sunk into” the background of practice. Because of this relationality, established infrastructure resurfaces once more in special moments, those of “infrastructural inversion,” either through the methods of a social scientist or as a result of a fault or breakdown which disturbs practice (Star 1999).

The WCS aspired to infrastructure. It aspired to become the basis for collaboration across a wide community of researchers. But becoming infrastructure is not straightforward and certainly hard to impose. Despite the best efforts of the team, WCS ended up being little used (Star 1999: 380). The problems it faced could not be traced to a single root cause; the ethnographers encountered diverse resistances cropping up across diverse sites. To make sense of these challenges, Star and Ruhleder turned to the cybernetician Gregory Bateson’s theory of communication, arguing that contextuality itself had frequently become the source of problems for the project. For instance, technical instructions that were seen as simple information about what to do by their originators, were for some recipients complex signs that differentiated kinds of persons: those for whom the instructions appeared straightforward, and those for whom they were anything but.

As with Bateson’s levels of communication or learning, the issues become less straightforward as contexts change. This is not an idealization process (i.e., they are not less material and more “mental”), nor even essentially one of scope (some widespread issues may be first order), but rather questions of context. Level one statements appear in our study: “Unix may be used to run WCS.” These statements are of a different character than a level two statement such as “A system developer may say Unix can be used here, but they don’t understand our support situation.” At the third level, the context widens to include theories of technical culture: “Unix users are evil—we are Mac people.” As these levels appear in developer-user communication, the nature of the gulfs between levels is important. (Star 1996: 117)
Successful infrastructures bridge local contexts. The process of *becoming infrastructure* is thus liable to act as an irritant for all kinds of contextual particularities. The connection I have in mind with configuration management, with Burgess and Promise Theory, is not about how they imagine infrastructure, but rather, this: the way in which, in circumstances of configuring complex distributed IT systems, locality becomes ontologically foregrounded.

To make the resonance clearer, the connection might be made with Star’s earlier and similarly influential research into cooperation. In her 1989 account of Berkeley’s Museum of Vertebrate Zoology, written with James Griesemer, she argued that consensus is not required for cooperation or for the emergence of common understandings. Cooperation, in short, must emerge across boundaries of intelligibility, rather than breaking those boundaries down. They argued, indeed, that “all science requires intersectional work” (Star and Griesemer 1989:392 emphasis added). Their concept of “boundary objects” denoted those interactive forms that mediate and coordinate across divides of intelligibility. These included *repositories* such as libraries or museums, *ideal types* that “delete local contingencies from the common object” (Star 2015, p254) such as representations found in scientific atlases, *terrain with coincident boundaries* such as the common referents on maps (while the professional biologists and amateur collectors Star and Griesemer studied produced very different kinds of maps, the referents they had in common facilitated their collaboration), and administrative *forms and labels* which use semiotic constraints to standardise information (Star and Griesemer 1989).

Star later complained that in its scholarly reception, the concept of boundary objects was most firmly associated with interpretive flexibility (Star 2010). Any such object would indeed need to be meaningful across different local contexts. But she was also concerned with the composition of open systems, something that received less attention. In an early formulation presented to an audience of artificial intelligence researchers, Star suggested that boundary objects are “simultaneously metaphor, model, and high level requirement for a distributed artificial intelligence system” (Star 2015 [1988]: 249). Such a system would need to be characterised by processes that mediate between their local particularities and higher-order coherence. For boundary objects, that meant a “tack[ing]
back and forth” between the kind of vagueness which renders things capable of being held in common between heterogeneous viewpoints and their local manifestations specific to just one (Star 2010: 604-605).

It would not be hard to read Promise Theory in these terms. Unlike deterministically imagined obedient computer networks, the coherence of smart intentional infrastructure emerges from just this kind of “tacking,” sailing against a prevailing wind (whether figured as “rot,” “drift” or “divergence”) by means of a series of zig-zagging trajectories. In common is the proper policy, the official record; in contrast its local manifestation is the contingent promise of the thing itself, which always of course may be otherwise than what it is supposed to be. The functional coherence of the whole is an achievement neither of the ideal or the contingent, but of the processes by which they are brought into interaction.

Star and Burgess, in other words, developed theories and pragmatics for building systems of distributed coordination. In both cases the promise of understanding emergent coordination across many, locally heterogeneous, nodes, required suppressing the intuition that this would be done, or explained by, the imposition of a single master ontology across the network, and in both cases the authors were engaged in projects for configuring systems, rather than acting as disengaged observers.

**Mythologies, or Why Figure Configurations?**

Promise Theory stands out among technical philosophy for its refusal to give priority to abstract, formal understandings of computation, typified for instance in the mathematical theory of algorithms, and instead emphasises the empirical materialisation of computing infrastructure, in specific networks and functional distributions, entailing contingent and localised embodiments of intent. The lack of a privileged centre was likewise a starting point for Star. She cites David and Smith: “When control is decentralized, no one node has a global view of all activities in the system; each node has a local view that includes information about only a subset of the tasks” (quoted in Star 2015 [1988], p246). A distributed system is brought together by means of boundary objects, not by obedience to a master command.
The tension between a formal “algorithmic” interpretation of computing and a more materially grounded approach is familiar and lively, perhaps never more so than in recent debates in critical studies of technology. For Ian Bogost, “the algorithm has taken on a particularly mythical role in our technology-obsessed era, one that has allowed it to wear the garb of divinity” (Bogost 2015: np). “In its ideological, mythic incarnation,” he argues, “the ideal algorithm is thought to be some flawless little trifle of lithe computer code, processing data into tapestry like a robotic silkworm. A perfect flower, elegant and pristine, simple and singular” (Bogost 2015: np). In the lexicon of media theory, as well as in general parlance, the concept of “the algorithm” has become a handle with which to grasp the implications of computing in society. Divine, we might surmise, because such an abstract understanding leaves little room for appreciating the locality of intent. A god has surely little need for boundary objects.

Allowing the formal abstractions of algorithms to stand as synecdoche for the material complexities of computational infrastructures is irresponsible (a point also argued by Chun 2011; Dourish 2016: 2). “Concepts like ‘algorithm’ have become sloppy shorthands, slang terms for the act of mistaking multipart complex systems for simple, singular ones” (Bogost 2015: np). Much the same sentiment is voiced by Burgess, who opens his book In Search of Certainty with the proclamation that “[t]he myth of the machine, that does exactly what we tell it, has come to an end” (Burgess 2015a: 1).

Bogost, in an echo of Marx’s analysis of “commodity fetishism,” suggests that the divine algorithm is falsely animated by a trick of the eye, by which we overlook the material conditions of production, the real work that goes in to creating these effects. Burgess suggests that our problem is our lack of a sufficient vocabulary to address the empirical locality associated with contingent technical systems. The mobile associations produced in the wake of the automation of configuration management do both jobs: the “engine,” the “phagocytes,” and the “promises” evoke the missing agency of technical systems, while the expansion of automation tools populates the world of IT configuration management with metaphors that give fresh form to the subject positions of the otherwise overlooked maintainers and repairers, whose hidden work, in the background, enabled computational systems to look like “robotic silkworms” in the
first place. “Puppet” dressed up the system administrator as the puppeteer in control, and “Chef” as one engaged in a finessed art of high esteem, both in stark contrast to the beleaguered service personnel in stereotypically subterranean offices, grappling with unwieldy infrastructures, mundane problems and an onslaught of helpdesk requests.

Metaphors stick and slip. Just as a controlling puppeteer hardly captures the autonomy that Promise Theory attributes to technical things, so too do well-worn metaphors come to stand for the opposite of their original intention. “Orchestration,” for instance, was one of the earliest metaphors for the automated management of networks. It suggests many autonomous parts moving in concert, but like the puppeteer it also implies a centre of control. Commentators were already using this connotation to tease out significant differences in the early 2000s. “Orchestration always represents control from one party’s perspective. This differs from choreography, which is more collaborative and allows each involved party to describe its part in the interaction” (Peltz 2003, p46). Kubernetes, probably today’s most well-known distributed computing system, is widely referred to as an orchestration system, yet reflects on the ambivalence of the metaphor within its own documentation: “The technical definition of orchestration is execution of a defined workflow: first do A, then B, then C. In contrast, Kubernetes comprises a set of independent, composable control processes that continuously drive the current state towards the provided desired state” (Kubernetes n.d.).

Metaphors of puppets and orchestras may appear blunt instruments to humanistic critics accustomed to searching for subtler layers of significance. But as figuration, as re(con)figuration, the working over of even these blunt instruments is an index of a question: what is (a)kin to the systems we are building and looking after? If figurations are inhabited, in the sense of being stances, ways of readying oneself in relation to another, attending to figurations might attune us to the ways that the evolving nature of technical stewardship has re(con)figured relations, and not just the words we use to talk about them. The stance we take towards distributed intentionality, towards our “smart intentional infrastructure,” is not just how we choose to represent it, but the kind of readiness entailed in relations to the local and contingent in a complex system.
The work of figuration nevertheless stirs up in its wake a detritus of metaphorical imprecision, and with it an enduring salience of “myth,” a treasured metaphor for metaphors gone stale: mistaken, naïve, of their moment. For what we do with “myth” is exactly what Star elicited through Bateson. By referencing Western stereotypes of “primitive” thought, myth takes our communication “up” a level or two. It draws attention from the content to the context of the “mythical” belief, as some Other’s belief, which sorts out kinds of persons, those provincial people, from elsewhere, or back then, who would take it at ground level. The kind of person who would be taken in, and who they are like. Applied across contexts, it dredges up infrastructures both entrenched and would-be: the perfect flower algorithms and the obedient computer.

The urgency of debunking myths in the critical studies of technology is a legacy of its obsession with the politics of representation, with the metaphors in and of technology, with “whose metaphor brings worlds together, and holds them there” (Star 1990: 52). If a newer wave of scholarship might be identified, around what Amoore calls a “cloud ethics … concerned with the political formation of relations to oneself and to others” (Amoore 2020: 7) “sustained by conditions of partiality and opacity” (Amoore 2020: 8), would it be a surprise if it were precisely practical relations with heterogeneous distributed computing systems that prompted the most radical re(con)figurations of the technical beings we live among and through?

Acknowledgements  Many thanks to Scott Wark, Celia Lury and Will Viney for their comments and feedback and for their efforts in bringing this collection together. This research was supported by the UKRI Future Leaders Fellowship Scaling Trust: An Anthropology of Cyber Security (MR/S037373/1). No new data were created in this study.

References


Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (http://creativecommons.org/licenses/by/4.0/), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter’s Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the chapter’s Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.