UNDERSTANDING INTENTIONAL SYSTEMS WITH RESILIENCE TO FAULTS, ERRORS, AND FLAWS

(DRAFT ONLY OF WORK IN PROGRESS)

UPDATES WILL BE POSTED AT HTTP://MARKBURGESS.ORG/FAULTS.PDF

Mark Burgess
markburgess.org

May 18, 2016

Disclaimer: Although he does his best, this work in progress promises only best-effort to formalize a mass of disparare experiences and inputs, shared in the hope that it helps to drive a conversation and shape the author's thoughts. It reserves the right to contain naivités, errors, or even things about which the author decides to change his capricious mind. Comments are always welcomed.

Acknowledgements: I am already grateful for comments and discussions by Adrian Cockcroft, Paul Borrill, as well as examples adapted from the experiences of a number of engineers in the field. None of these bear any responsibilty for errors, omissions, lack of clarity, or interpretations contained herein.

Contents

Abstract

These notes assemble a set of definitions, concepts, and techniques, for a scale dependent analysis of reliability in cooperative systems. This treatment is based on the language and methods of promise theory. It extends classical statistical analysis of component reliability to deal with context dependence and non-trivial functional semantics. The active agents in a system (humans, machines, etc) are generic; they are assessed only by their accuracy or *fidelity* in keeping to their promised roles. Adaptation and evolution of systems fall outside the scope of this introduction, except in the most primitive contextual forms.

Promise theory focuses on describing intended outcomes, within these systems. This forms a basis for qualitatively and quantitatively assessing partial systems, with suitably idealized approximations, based on calibrated measurement, and semantic reasoning.

Chapter 1

On the reliability of functional systems

> "As robotic system developers strive to achieve a level of autonomy, they underestimate the need for coordination with human stakeholders." [DH06]
>
> "The most powerful dehumanizing machine is not technology but the social machine, i.e. The formation of command structures to make humans emulate technology in order to build pyramids and skyscrapers..."
>
> –Lewis Mumford (1967)

1.1 Manifesto

Developing formal models to describe resilient or reliable systems is a singular challenge. Systems are inherently subjective interpretations of dynamical processes, and treating subjectivity, in as impartial a manner as possible, presents several obstacles. Nevertheless, this is the challenge of any systems theory, and it is what promise theory was created for.

Classical reliability theory, following a mathematical tradition, has a history of approaching the concept of manufacturing reliability statistically[Nat98, HR94]. Systems were assumed to be networks of black box components, and faults are treated as statistical occurrences within the components rather than their interactions. Moreover, the functional semantics of components were suppressed deliberately to give an impartial account of faults that was generic but limited. To modernize and complete a model of functional systems, in which semantics play a major roles (e.g. interactive systems, software, and services[Cha05]), one is forced to involve systemic issues at multiple scales, as well as 'human issues' (all semantics have human origin)[HWL06, DH06, Cil98, Bur04a].

The aim of this approach is to capture some of both these views, by exposing the semantics and modelling the functionality along side the fidelity of active and passive elements. Trying to describe systems without breaking them into interacting parts leads only to vague descriptions that emphasize subjectivity. By using promise theory to describe systems through agent interactions, we can find a useful middle ground that respects the traditions of physics, and places subjectivities on a clear footing.

Faults and errors involve deviations in semantics and interpretation as well as deviations in dynamical behaviour. The aim of formalization is not to present the perfect *fait d'accomplit*, but to bring some straightforward enhancements to the state of system formalization.

1.2 Systems

A system begins with assembly of component parts that interact and develop. Unless trivial, systems exhibit bulk behaviours that cannot be fully understood from the behaviours of its components in isolation. Any assembly of parts, collected within any arbitrary boundary, may be viewed as a system. Such phenomena exist in nature, or they may be artificial.

Definition 1 (System) *Any set of component resources, their interactions, and patterns of change, that can be assembled into a descriptive model with associated semantics.*

The ability to describe a system is important: without this, we cannot speak of a system at all. Systems are always identified by human observers, which means that they are often based on arbitrary choices, and this affects the nature of what we see in them and mean by them. Subjectivity is a key aspect of systems, independently of how we might strive for impartiality in describing them.

1.2.1 System boundaries, open and closed systems

If a system were completely decoupled from its surroundings, and its users, we would not even be able to observe it or interact with it. So where does 'the system' start and end? Does it include users, its environment? Surely both of these interact with it, and therefore influence it. This is where system design often becomes rather wooly. One tends to neglect the users and the environment in the interests of modularity, but, although convenient, this is paradoxical if not slightly dishonest (see figure 1.1). There has to be a way to focusing on the most relevant pieces of a problem, and approximating away the rest. This points us to the notion of a 'suitably idealized approximation' in science, in which we define subsystems as 'the system', from the viewpoint of an observer. There are well established techniques in science for making such approximations consistently.

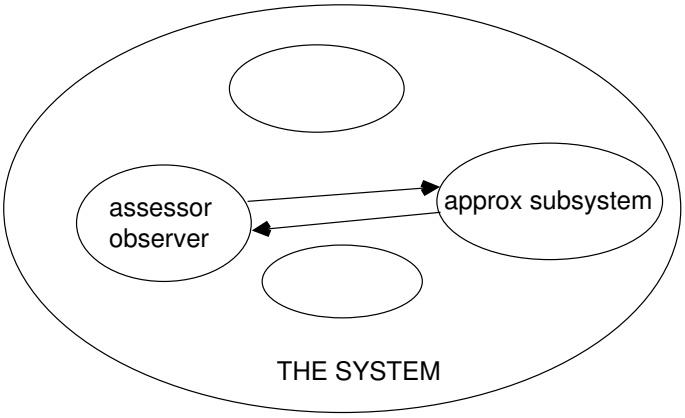

Figure 1.1: Where is the edge of a system? We should really talk about embedded subsystems, as no system has a meaningful edge if it interacts with a party defined to be outside of itself.

The system is everything that is connected through interactions. If we choose a random boundary to define a subsystem of interest the results might be surprising, because interactions across the boundary can play a major role in the outcomes within the boundary. So we must try to define boundaries sensibly as regions where the *coupling* between what is inside and outside the boundary is weak, or even absent. Interaction is key to understanding system behaviour.

Following the traditons of physics, we may begin by defining open and closed systems:

Definition 2 (Open system) *An arbitrary collection of interacting parts (called agents), each of which may interact with parts not defined to be within the system. Open systems are really subsystems with locally and globally incomplete information.*

Definition 3 (Closed system) *A closed collection of parts (called agents) each of which may interact with other parts inside the collection. Closed systems are total systems with globally complete information.*

Closed systems are idealizations and approximations to reality. They are an artifice, defined to eludicate specific issues in a model, such as the formulation of simple rules. All practical systems are open systems.

Comment 1 (Top-down versus bottom-up) *Many engineers (especially in computer science and engineering) are taught to design systems from the top down: that means starting with the whole and breaking it into successively smaller component parts. The problem with this methodology is: we can't build a system from the top down (except perhaps a sculptur who whittles away parts), so designing it this way leads to a mismatch between intent and capability.*

Top-down is a narrative construction: it is the way we explain through stories. It assumes to know what the top of the system is; however, we have already said that the boundary of the system is arbitrary. So what, then, are we designing? If we design from established components, that can make known promises (materials, off the shelf parts, etc), and build up, then we have an open ended process with a stable foundation. If we build from an uncertain 'top' towards an uncertain foundation, the whole edifice of the system is uncertain: nothing may be stable.

1.2.2 The purpose of a system (semantics)

Whether a system is natural or artificial in origin, we may describe it as having a *purpose* if 'observers', 'assessors', or 'users', who interact with it, project semantics onto its collective behaviours. Each assess it in relation to their personal set of understandings and sense of values. This is an inherently subjective matter, which makes assessing system outcomes itself controversial. The quotes at the start of the chapter illustrate a few of the pitfalls and preconceptions that await the unwary, in attempting to discuss the idea of failures and errors in systems.

In a system that was evolved rather than designed, there is no explicitly promised purpose. Evolution is merely a competition between mutation and constrained selection, and can only describe a purpose indirectly by ascribing a meaning to the selection criterion. In nature, the criterion is stablity within a particular niche context, i.e. niche survival. Artificial evolution may be based on design goals, or choices, e.g. genetic algorithms. In this case, the promises are implicit, and lie in the value judgement of observed behaviour.

1.2.3 System behaviours (dynamics)

The observable outcomes of a system are measurable states, defined at different scales of observation[Bur15]. It is assumed that a system may be described by the propagation or evolution of such states over time and space[Bur14]. In fact, a timeline is defined to be a sequence of changes in the states of the system! This kind of model for 'propagation' from one state to another is well understood in physics.

Definition 4 (State of an agent (microstate)) *A description of the variables that characterize the condition and activity of the agents, at a given scale of observation, along with their rate or probability of change (trajectory)*[a].

[a] In continuous systems, the state is the position and generalized velocity. In a discrete system, the state is the position and the transition probabilities.

Definition 5 (State of the system (macrostate)) *A description of the bulk average observable properties of the system, at the scale of observation.*

Note how the scale of observation plays a central role in describing the system. This reflects the fact that we always have limited, and generally incomplete, information about a system. In other words, our knowledge of a system is always some kind of measured approximation to what is going on.

- Outcomes are states that are the results of interactions between agencies of the system. Interactions consist of two kinds of promise, labelled + (offered or intended) and - (accepted or received).
- Behaviours may be characterized by observers viewing at different scales.
- The strength of agent-agent interactions determines how well behaviours at different scales (de)couple.
- Strong interactions may be considered hard dependencies that form a network along the interaction paths.

Lemma 1 (Open systems have unpredictable states and trajectories) *We cannot predict all behaviours within an open system, as new information can enter that is, by definition, not predictable. PROOF...*

In a closed system we have, in principle, complete information. Of course, that assumes that the system is also closed from within, i.e. that no unexpected behaviours come from within agents in a system (such as breakages, capricious intent, or even quantum mechanical uncertainty). The notion of a closed system is essentially an idealization to help isolate certain causal trajectories in system dynamics, and enable formalization. They should not be considered realistic in the world around us.

1.2.4 Super-agency and scaling

We may aggregate a collection of cooperating agents into an aggregate 'super-agent', that behaves as a single entity at a larger scale. Promises that only affect what is inside the super-agent may be called interior, and promises that affect what is outside may be called exterior (see figure 1.2). In this way, we can imagine partitioning the entire world into open subsystems that we define in any convenient manner.

1.2.5 Predictability and promise

If a system is stable enough to be predictable, then we can describe its patterns of time development according to some formulae[1]. We may only be able to describe the envelope of possible outcomes by a formula, or we might be able to predict the exact state of the approximate model! Promise theory allows both dynamic and semantic outcomes to be placed on a similar footing[Bur13] by defining comparative scales for outcomes, which we call promises. This is a useful approach, with great flexibility.

1.3 A human bias, and scaling

A common error of judgement that designers make about systems lies in assuming that systems work in the way that humans work: by exerting their will as if a system were the extension of our bodies. It affects the way we design systems, and the way we judge and interact with them, and thus its reach into our expectations for system behaviour is long and subtle. It boils down to a choice between two extreme forms of organization and control that I like to call *brain models* versus *society models*.

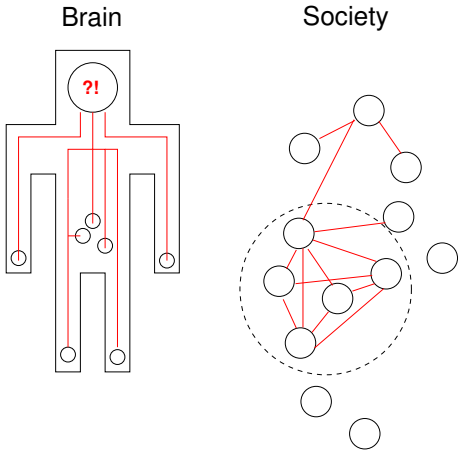

Figure 1.2: Brains versus societies. A brain model imagines a definite edge to a system. In a society model, the scaling is more arbitrary, but we can speak of super-agents with arbitrary boundaries, e.g. the dotted ring.

A *brain model* has a central controller that receives data from sensors and imposes commands onto actuators from its central command (see fig. 1.2) out to the skin, sensors, and actuators of a finite organism. A *society model*, on the other hand, has no particular centre, and no particular edge. It works by cooperation on an agent-by-agent basis. This is how insect colonies work.

[1] Some like to use the term rules or laws of motion, but this suggests that they have some absolute nature, which is not the case. The baheviours are emergent, but might be sufficiently repeatable to warrant capturing as a formula.

Brain models are centralized concentrations of capability, used to drive a larger system from a localized component. Although I use the analogy of a brain, hearts and livers, and other organs also behave in this way, though they do not use the full potential of the brain model. Brains coordinate information quickly and analyse it in ways that societies can't, because of the close proximity of agencies with different specializations, and an assumption that they work faster than the systems they are controlling. However, this means that brains scale by brute force to handle inputs and outputs. Both total load of information, and the distance information has to travel play a role in limiting the scalability of control processes. The largest organism with a brain is a blue whale. It is quite slow. Indeed, the larger organisms with brains get, the slower they behave, because the costs of communication and processing become an eventual burden. Insects, on the other hand, react without centralization, based only on what policies they have cached in their DNA (zero distance to travel). Their colonies scale to much larger size, before food (energy) becomes a limiting factor, but they can't easily perform the same kinds of analyses and decision making when they are sparse[2]. These are the kinds of scales and processes we need to formalize and understand quantitatively to develop a proper understanding of systems.

We often expect that, by willing systems to behave as we would like them to, they ought to comply, because we are used to that mode of cause and effect in which we have an advantage of being able to observe and think faster than the system we are controlling. Ants or societies, on the other hand, do not work like that. Causation is subtle, and travels as if by Chinese whispers. The network has to converge towars some equilibrium at a single rate, without an external force to modulate it. What they can do with their tiny brains and simple manipulators is quite trivial, and yet the emergent behaviour of thousands or millions of ants leads to many stable and interesting outcomes.

Brain control is limited by surrounding system size, speed of thought, and speed of communication with surroundings. It is an arms race in speed. Societal control is limited by power consumption and communication. Both are limited by stability. We like brain models because that's what we know. We don't like societal models because they are hard to understand and they don't work from a position of superiority, but in practice we need both to understand any sizable system.

1.4 The role of the observer

How a system looks, and what it does, depends on how and where you look at it. It is a subjective sampling of the world. In any distributed system, we are forced to confront such subjectivities. Our aim is to do this in impartial way, using formal language that can be clear. This is one of the aims of Promise Theory[BB14].

A system with a set of clear promises can use these as a calibrating 'measuring stick'. One can measure both qualitative and quantitative aspects of its behaviour against this set of promises. However, the point is not that promises are always kept. systems of people and machinery do not always work as planned. Outcomes can be thwarted in three broadly defined ways:

- Mistakes in the intended execution of a plan (errors).
- Unforeseen interruptions to behaviours that we did not plan for (faults).
- Flaws in the plan itself (ill conceived plan or changing our minds).

These three axes are not completely independent (some will even argue that it is impossible to define these at all), but I shall assume that they form a useful basis for approximating the issues and building models.

1.5 The use of promise theory as a perspective

It is important to have a framework in which to formulate a model, with clear definitions the reduce unnecessary arguments about terminology. Promise Theory is a convenient abstraction for describing the scaling of intent. If an agent, behaving within a system, makes no promises, then it cannot be in error, nor can it exibit a fault, even if it fails to meet others expectations. Thus, all components need to document their intended functions and limits.

[2] It might or might not be true that a brain works somewhat like an ant colony, but it serves an organism at a larger and slower scale than its own interations. An insect colony cannot think like a brain at the same scale any more than a brain can analyse and rewire its own behaviour in real time, because learning itself implies the involvement of a scale that is an aggregation of the basic causal processes.

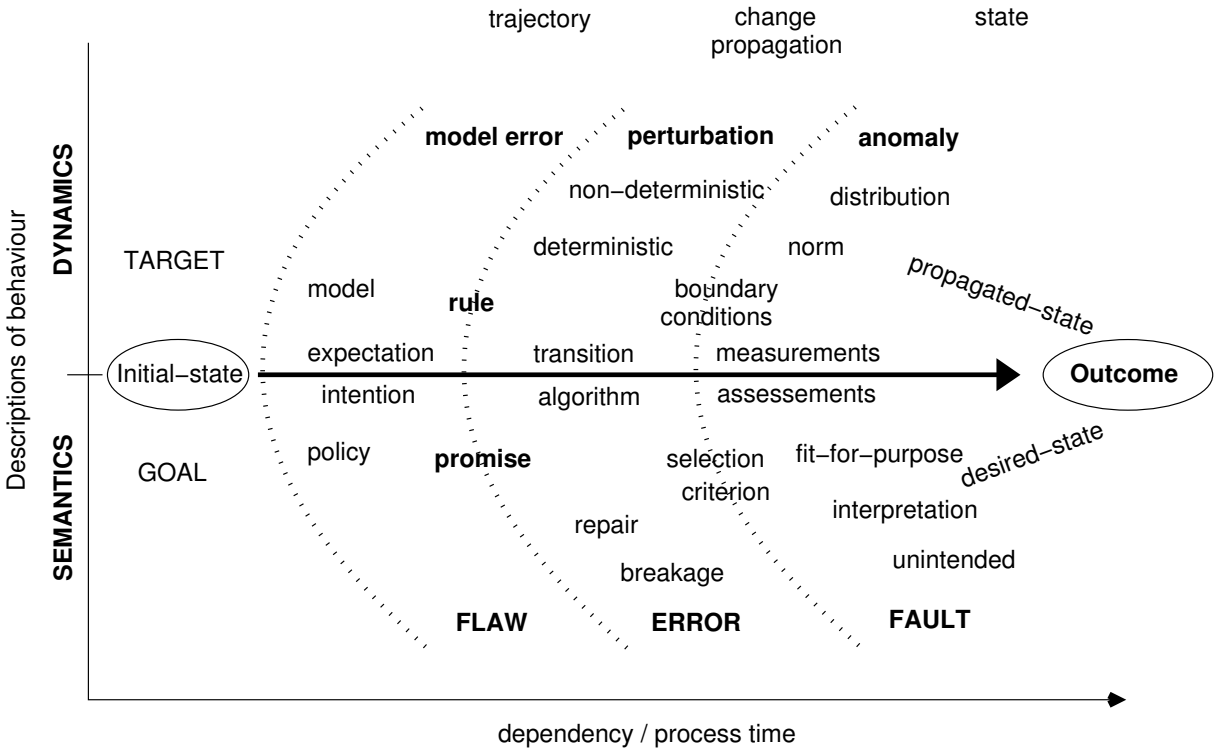

Figure 1.3: A mind map of descriptive causation, or the prediction lifecycle, comparing dynamics and semantics by analogy. There is a vast amount of related terminology in use. Predictability follows the arrow. Drift takes us into the realm of unwanted outcomes.

The basics of Promise Theory are described here:

1. All states, actions and behaviours in a system are properties of agents.
2. Without intent, an agent's behaviouris said to be *ad hoc* or unconstrained.
3. Without coordination intent, a collection of agents cannot act in an intentional way, and its collective behaviour must be considered *autonomous*.
4. Agents are a priori atomic, i.e. independent or autonomous[3]. They may promise to give up some or all of their independence by receiving input from others.
5. Agents make promises, and assess the promises made by other agents.
6. Intentions depend on *context*, i.e. the states reported by other agents within a collaborative system. Thus agents may be able to assess context and promise to share that assessment transparently as a service to other collaborating agents.
7. A documented intention is called a promise.
    - It is made *by* the agent that has the intention and can keep the promise.
    - It is made *to* any stakeholders with an active interest in the information, and may be seen by others too with only passive interest.
    - No agent can make a promise for another agent to keep (that is called an *imposition*).
    - In a system, it is not sufficient that each agent knows its own intent: intentions may have to be shared with others, so that agents can form expectations about one another's behaviours. This is how systems work together in a collective or collaborative manner.

[3]This does not mean that they can necessarily work independently of external help, only they can always choose to ignore it if they so wish.

8. A promise does not guarantee an outcome. The promised outcome may be kept with 'best effort' by the agent concerned. An outcome that is not promised may be considered *ad hoc* or *taken for granted*. No agent in the system can form any expectation of a non-promised outcome.

9. An agent *policy* is a collection of promises that attach to assessable contexts.

10. A promise that does not promise a final state (e.g. only promises a relative transition), and which is independent of its starting state, is unstable and cannot be used to predict outcomes over time.

11. A system in which each agent does not promise a coordinated intent cannot collectively promise a desired outcome, and a user cannot expect anything about its behaviour.

Context sharing leads to greater information and a more nuanced selection of promises approprate to adapting to situations that arise. The desired-state stability of promises is what can prevent this adaptation from becoming chaotic.

Comment 2 (Documented systems: situation awareness) *There is a case for systems that can explain themselves to themselves as well as to others. Such systems expose their promises, enabling other agents to form expectations and adapt. A system that makes clear promises allows observers to be aware of the state of the system.*

Very few systems are designed and documented in this way. Even worse, systems do not promise how they behave under all conditions and fail to behave predictably, because agents are unaware of either their intent or their behaviour or both. If technology is designed with clear promises, and an explanation of how those promises are kept (at all times), then it becomes self-documenting.

Systems are commonly measured ad hoc*, without a guiding scale of promises or expectations. In this case, we simply don't know what we are measuring or why. The concept of a Key Performance Indicator is like a promise assessment. Compare this to a random act of monitoring, such are systems.*

Comment 3 (Taking for granted) *We 'take things for granted' when no promise of a behaviour has been made, but we make use of the behaviour anyway. Promises are expensive to make and to keep, so it is statistically likely that the amount of information in our expectations exceeds the amount of information covered by our promises, hence there is a statistical likelihood for faults to arise through incomplete documentation of intent (specification).*

Such systems work by 'take it or leave it' behaviours, and the reponsibility is usually left to human operators to sort of the resulting mess. This in turn leads to the generic conclusion of 'human error', as the result of a cultural heritage of treating humans as machines.

It is useful to add a definition of a system design:

Definition 6 (Design) *The design of a system consists of an inventory of agencies and all the promises made, for each identifiable context, both from within and without, i.e. including at the system boundary with 'environment' and 'user'.*

A design that encompasses all agents and all promises may be called complete, else it is incomplete.

A design's success in keeping its promises may be evaluated to talk about its fitness for purpose. Proving completeness is a key issue, but it is generally impossible, as systems are open.

1.6 Defining plausible semantics for system anomalies

In terms of the formalized view of promises, we can ascribe reasonable meanings to the terms, under the general heading of anomalies.

1.6.1 Errors (of execution)

An error is a term that applies to processes or actions, not to outcomes.

Definition 7 (Error (of execution)) *An* action *or change whose outcome is assessed not to comply with the promises it makes, by some agent.*

There must be a promise, as a precondition, in order for there to be an error. The promise defines an desired outcome, and plants a measuring stick for the outcome of the promise to be assessed by some observer. An error of execution may or may not lead to a fault condition, depending on the resilience of the system and the degree of detail to which it is inspected by assessing agencies.

The theory of measurement errors[Top72, Die02] distinguishes between two types of error:

- *Random errors*: Unpredictable variations in the keeping of measurement promises.
- *Systemic errors*: predictable biases distorting the keeping of promises. These are often related to miscalibration of observation, relativity and misalignment of understanding, by an observer, therefore we cannot deal with these in an impartial way.

It is known that human-related variations are often Gaussian, or normally distributed, but other system influences lead to variations that are often asymmetrically distributed[BHRS01].

1.6.2 Agent accuracy or fidelity

From the notion of an error, we can define a general term for the reliability of an agent, in carrying out actions related to promise keeping.

Definition 8 (Agent fidelity (accuracy)) *The degree to which an agent performs actions predictably, and in keeping with promises, i.e. without error.*

In this form, it no longer matters whether the role is played by a human of a machine: an agent (whether human or machine) that makes frequent errors may be said to act with *low fidelity*, as it is not able to keep its promises, even though it intends to. Moreover, by grounding the assessment with a specific promise, we help to eliminate room for speculation about intended outcomes.

1.6.3 Flaws of design (fitness for purpose)

We must also consider the notion that a promise itself was ill conceived, i.e. it was assessed to be of low value, relative to the goals of the system.

Definition 9 (Design flaw) *A promise that is assessed to be inappropriate for the intended outcome, mistaken in its aims, or misaligned with collective goals, by some agent.*

We sometimes talk about a system being 'fit for purpose'. This does not necessarily mean that its intent was misplaced, but that the model of promises was incomplete or insufficient to cover the dominant expectations and eventualities.

Comment 4 (Expecting too much) *With the exception of errors of execution, which we may ascribe to low fidelity of component agents, errors occur often for the simple reason that our expectations exceed the quality of our intentions. In other words, we tend only to form appropriately detailed expectations about what 'should have been intended' only after the horse has bolted (*post hoc*), when it is too late. Thus we can still learn from failure.*

As systems operate, in unpredictable environments, the number of possible states they can occupy grows, by interaction with the environment, to areas where agents have not made promises. Thus agents find themselves without guidance. In other words, systems are incompletely specified, or work on incomplete information.

1.6.4 Faults (anomalous states)

Faults are *conditions* or *states* within a system that do not match our expectations, or comply with their promise.

Definition 10 (Fault) *A state or condition, in which the system does not match the promised or desired state. This leads to unfulfilled expectations. In other words, a condition in which an assessment (sample) of the system fails to keep its promises.*

A fault may be arrived at by an error of execution, or in the absence of error because of a design flaw. We might never make something truly fault free, because design flaws can always be hard to foresee. There is thus a strong practical need to build systems that are *fault tolerant*, rather than fault free.

Definition 11 (Fault tolerance) *An agent may be called tolerant if it continues to keep its promises, within acceptable tolerances, even though it depends on promises from other agents, that may or may not be kept.*

Note that, once again, without a prescribed set of promises for an intended state, there is nothing to define when a fault has occurred[4].

Example 1 (Buffers for absorbing variation) *Faults propagate when agents that depend on them are affected by them. Buffers are a way for agents to absorb variations. Buffers are used in mechanical systems (fenders, air cushions, etc), in financial systems (cash flow buffers for unexpected expenses), in allowed leeway for systems in motion, and so on.*

The occurrence of system faults is an extensive and involved topic that is the subject of whole texts (see, for instance [Nat98], [HR94], [NRC81] and [SS03], [DH06]). System faults fall into three main categories, by source:

- *Random faults*: unpredictable occurrences or freaks of nature, usually distributed in a random fashion, leading to an incorrect outcome.
- *Emergent faults*: the system exhibits semantics that it was not designed to promise. These usually come about once a system is in contact with an environment, or through intra-networking at scale.
- *Systemic faults*: faults which are repeatedly caused by logical flaws of design, or insufficient specification.

Comment 5 (IEEE standard anomalies) *The IEEE classification of computer software anomalies ([IEE]) includes the following issues: operating system crash, program hang-up, program crash, input problem, output problem, failed required performance, perceived total failure, system error message, service degraded, wrong output, no output. This classification touches on a variety of themes all of which might plague the interaction between users and an operating system. Some of these issues encroach on the area of performance tuning, e.g. service degraded. Performance tuning is certainly related to the issue of availability of network services and thus this is a part of system administration. However performance tuning vis of only peripheral importance compared to the matter of possible complete failure.*

Many of the problems associated with system administration and maintenance can be attributed to input problems (incorrect or inappropriate configuration) and failed performance through loss of resources. Unlike many software situations these are not problems which can be eliminated by re-evaluating individual software components. In system administration the problems are partly social and partly due to the cooperative nature of the many interaction software components. The unpredictability of operating systems is dominated by these issues.

1.6.5 Discoveries (classification anomalies)

Does this cover everything? No: what if we have not had the foresight to even make a promise, because we weren't expecting an issue? In this case an outcome is merely something that we identify ad hoc, with a lack of understanding or incomplete information about the system. We could call the observed effects, events, anomalies, surprises, or even discoveries of new phenomena.

[4] Hand-waving heuristics or 'bad feelings' are often used to cry foul. Bad feelings are not very useful for engineering, but they cannot be dismissed either. Human intuition is mysterious, and can often sense 'danger' without being able to form a narrative to explain it.

Definition 12 (Phenomenon discovery) *An state of a system that was unexpected because no promise was made by the affected agency concerning the observed outcome.*

Surprises, like this, might not be possible to measure directly, since there is no *a priori* basis by which to describe it.

1.6.6 The usefulness of these definitions: the matter of design

We have the terms:

- Errors for actions that were not carried out as promised (transition anomalies).
- Faults for states we arrive at that were not as promised (state anomalies).
- Flaws for the suitability of the intention itself (assessment anomalies).
- Discoveries for unexpected occurrences or outcomes that do not match any promise (classification anomalies).

Having rigid definitions of intent makes it easy to follow a formal method, even if the definitions are not perfect. With the definitions above, we can:

- Push all of the ambiguity around the *source* of flaws onto a single idea: the 'design' of a system that reflects its fitness for purpose. This allows for systems that are 'complex', where causation cannot be attributed to any single element or agency.
- Push all of the ambiguity about faults onto the keeping of promises by one or more agents. We do not have to prejudge the reason why a promise was not kept.
- Push all of the ambiguity about error onto the fidelity of the agents within the system. Errors may or may not be considered the 'cause' of faults[5].

This is a highly pragmatic separation of concerns. We focus attention on issues by saying what we mean with promises. With these definitions, we are better equipped to answer these questions:

- What kinds of conditions can arise within a system?
- Are they desirable?
- If yes or no, were they intended, i.e. the results of errors?
- How could undesirable states be prevented or tolerated?

Key to answering this question is another question: how much coverage of a system's behaviour can be documented or promised? In other words, how much confidence do we have in our ability to determine the outcome? Conversely, how much is simply *ad hoc*, or even beyond control?

1.7 Instability and the limits of promises

Stability is the perhaps the single most important notion in reliability engineering. No matter how simple or complex a system is in its makeup, the key to predicting outcomes is its stability. The abilty for a system of any size to represent something consistently depends on its stability.

Definition 13 (Dynamic stability) *The insensitivity of a promised outcome to perturbations from any source.*

Definition 14 (Semantic stability) *The insensitivity of a promised outcome to variance in interpretation from any source.*

[5] While there is low level causation at work in the system, it is not necessarily traceable as a convenient human narrative of events.

Our ability to determine outcomes within a system can only be related to the promises it makes only if it is both sufficiently isolated from external couplings, and it is stable to perturbations through the remaining couplings. There are essentially two cases we have to consider:

- A system is sufficiently stable for promises to be kept most of the time, and may be used in a functional role.
- A system is not sufficiently stable and cannot be assigned any reliable meaning.

In the latter case, making promises seems to be futile, but there is still a role for them: even though promises can't be kept reliably, they define a basis set of outcomes against which to measure and compare the system's behaviour.

Sometimes instability is couched as a 'complexity' issue. In so-called 'complex systems', non-linearities introduced through strong couplings (dependencies) amplify small deviations, causing divergent or unpredictable behaviour. One can try to constrain or protect against this divergent behaviour, but it becomes a competitive game of information and speed[6]. It is unclear whether complexity itself necessarily plays a role in disallowing stable outcomes; however, complex systems are likely to probe states that were not anticipated or planned for, and hence one could not easily expect that promises about them would be kept.

The existence of a documented promise enable any agent, in principle, to probe and detect when this is happening. If single promises cannot be kept, over time, the source of trouble is easily localized to the agent making the promise, and its dependencies. On the other hand, a situation in which no promises can be kept (i.e. one is constantly having to repair state) is likely a situation of major instability, at the system level, not merely bad luck in a dud component.

1.7.1 Could all promises be kept and still yield unpredictable outcomes?

Promises are not guarantees, and they may not cover all possible outcomes. Could all of the promises offered to explain a system's behaviours be kept, but still leave room for inexplicable outcomes? The answer would seem to be yes, because we can only make promises about scenarios we anticipate.

A promise could be so ambiguously or vaguely formulated that it is actually worthless, because its outcome cannot be assessed e.g. 'We promise to protect the public from harm'. How shall we measure this? A 'dumb agent' may promise to follow a set of rules to the letter of the law, but fail to fulfil its intended goal: 'We followed the regulations but people still died'. In this case, the intermediary step of promising to follow impositional rules adds to the ineffectiveness of the promises; it is easier to assess when agents promise what they will do, rather than what they won't do.

Consider the following examples.

Example 2 *Imagine a 3 dimensional system that you measure only in a two dimensional slice: you will be constantly surprised by what seems to pop out of thin air. Many systems of agents make promises collectively, as if they were a single entity[Bur15]. For example, a radio, or television makes product promises about delivering entertainment, while the individual components promise electrical properties. A box of Corn Flakes makes a broad marketing promise from the box, representing everything inside it, while the individual flakes make quite different promses.*

Example 3 *A car is a collection of parts all of which might keep their promises. The sum of these promises does not imply the promise made by the whole, i.e. to be a car. Could all of these promises be kept, and yet still not yield the outcome of transportation? Imagine if the car is lifted off the ground by a crane, none of the promises are violated, but it does not succeed in fulfilling its function of motive transport. In this case, the reason is that there is a tacit assumption that the car will be on a road, yet this coupling to the environment is crucial. This is an example of overlooking something that is too obvious to mention (though tyre manufacturers might disagree).*

Promises refer to qualities that are either assumed to be under our partially deterministic control, or stable and repeatable, in spite of noise. The challenge we face, then, is what to do about the aspects of systems that we don't know about, and cannot make any kind of promise about. Systems of incomplete information are the potentially risky.

[6]The fighter jet is an example of this. In order to fly with great agility it is basically designed to be unstable. It remains in control by correcting for these instabilities with very fast computer automation. This is a case of very clever design around the edge of instability.

Law 1 (Fitness for purpose and incomplete information) *A system of agents may exhibit no faults, and no errors but still have unexpected outcomes.*

In this case, one may only argue that the system is *flawed* in its (possibly designed) fitness for assumed purpose, because insufficient promises were proposed to cover the outcome. The proof is simple: imagine a system that makes no promise (as in evolutionary systems without clear selection criteria). Such a system cannot have faults or errors, by definition, as there is no standard to measure against. In this case every outcome is unexpected, for the same reason.

1.7.2 Catastrophes, epidemics, and critical phenomena

Because interaction is the source of dependency in systems, topology is a significant factor in promise keeping. Highly connected agents become hubs for epidemic spreading of failure.

Definition 15 (Single point of failure) *Any agent within a system whose failure to keep a promise would result in immediate propagation of a fault throughout the system, i.e. the failure of external system promises.*

A single point of failure traditionally means a special node, which, if removed or disabled, would cause a complete stoppage in the functioning of the system. The definition in terms of promises becomes more precise. Thus we have the associated notion of mission criticality:

Definition 16 (Mission critical (sub)systems) *A subsystem that is a hard dependency within the larger system, and whose failure would result in an immediate negative consequences.*

The aim in any mission critical system is for any fault to have a manageable impact on the system, leading to (at worst) a temporary *degradation* of service rather than *interruption* of service.

Propensity for fault propagation is another effect that is understood from network science. Network *centrality* is a characteristic of an agent that measures its potential for propagating influence to other agents. *Percolation* is another critical phenomenon, whereby a system attains a critical density of dependencies such that it becomes possible for information (including faults) to travel all the way through a system from end to end.

1.7.3 Intrinsic stability: convergent outcomes, or policy model enforcement

Instability is associated with non-linearity, or amplification of effect. The key property for intrinsic stability, however, is not whether or not a perturbation of a system leads to a predictable outcome once, under particular circumstances, but whether the outcome is both reproducible and within tolerances. For this, even determinism is not necessarily enough. We also need *convergence* (see figure 1.4). Convergence to a so-called *fixed point* is a form of stability[Bur04a].

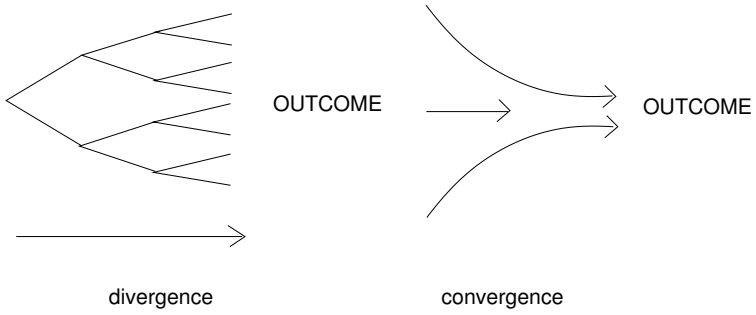

Figure 1.4: Divergence is the transition from state where the system is concentrated around a small number of states, to a large number of states. Convergence is the opposite: a concentration around a focused intended end state.

Convergence of outcomes is a sublinear response. This is the antidote to instability and variability.

Example 4 *Think of an explosion. An explosion is caused by a deterministic set of promises being kept between molecular agents that are brought together. The system is both unstable because the outcome grows exponentially in response to the perturbation of being 'lit', and yet it develops entirely deterministically. The result is that there is too much information arriving too fast to process and contain, hence the system is dominated by the explosion and the normal semantics are wiped out. We can only associate stable semantics with convergent outcomes that are of finite scope (i.e. local in some sense).*

The lesson of this section is that systems, which are designed by an intentional process, have to be designed so as to isolate and promise requisite stability, if they are to have predictable outcomes that users can rely on. That which is not promised, with the trustworthy intent to keep promises, is not to be depended on[7].

1.8 Agent responsibility, causal memory, and 'human error'

It is often said that human error[Rea90, Dek06, D$\ddot{8}$9] accounts for the bulk of failures and outages. This is an empty statement, because ultimately all decisions or oversights can be traced back to a human, making humans an easy default target. By abstracting both humans and machines proxies simply as 'agents', which make promises, we shift from a potentially prejudicial account of individual capabilities (and indeed liabilities), to impartial *roles*, together with an assessment how well they are performed.

In trying to attribute blame, we have to understand that:

- Agents may or may not be acting with complete agency, unconstrained by circumstances beyond their control.
- The effect or 'memory' of an action taken by an agent does not last forever, because system interactions mix information sources reducing their impact; so, there is time limit on how long one might be considered responsible for a decision or action. Complex (non-linear) systems 'forget' the information that was input at some time more quickly than linear ones. Influence of an agent is scaled down by mixing, and eventually becomes dominated by new factors.

Both of these apply to humans and machines alike. The consequence is that no single agent, whether human or machine, has a decisive or lasting effect on total system behaviour, in general. The impact of an agent on outcomes has to be analysed on a case by case basis.

From a promise theory perspective, whether an agent is human or not is relevant only insofar as it implies certain capabilities to keep promises. If one is impartial, then ultimately all that matters is whether promises were kept or not. Thus promises allow us to maintain impartiality while incorporating more subjective and qualitative issues into system description.

1.9 Agent accuracy (fidelity) - faults in human-computer systems

The accuracy with which agents are able to keep promises may depend on many factors, but ultimately such factors do not matter to the outcomes. This is why we are able to automate human processes using machinery. As long as they keep the same core of promises, surrogates and proxies may be exchanged for human labour. Most processes start out as manual human interventions, but given a sufficient understanding of the important promises, system agents can often be replaced by proxies that make the same promises, allowing humans to step aside.

Comment 6 (Humans, step aside and let the system prosper) *The focus on outcomes rather than imperative procedures, implicit in a promise viewpoint, means that one may optimize promise keeping without trying to imitate the means of implementation that would be suitable for humans.*

[7] The expression a 'black swan' event is sometimes used for rare unimagined occurrences. In this analogy, you wouldn't bother to promise that a swan is white, because it is so obvious.

1.9.1 Accuracy of active components (intentional agents)

It is tempting to delve into agents' characteristics, as we think anthropomorphically. Machines tend to offer better consistency (fidelity) in pursuit of simple unambiguous tasks, but not necessarily higher correctness, since correctness depends on the design's ability to behave in advancement of the system's major goal, in all contexts, something that requires awareness. Humans are currently superior in adapting to contexts (expected and unexpected). Machines do not notice changes of context unless they have been explicitly designed to do so; also, they are not aware of the consequences of their actions. However, we do not need to make any of these points, if we have promised outcomes in sufficient detail. The nature of agents is in the realm of speculation.

- **High fidelity agents** H_i: agents which keep their promises consistently, and with a high level of certainty.
- **Low fidelity agents** L_i: agents which fail to keep promises reliably.

The tautological nature of these definitions shows that we cannot assume that machines will perform better than humans. Such an assessment depends on many factors, e.g.

- Dumb machines are likely to be successful when isolated from unpredictability. *Machines are often act with high accuracy/fidelity during simple repetitive tasks, without adaptation. Humans may be high fidelity problem solvers, and designers, when working in short durations, within their field of expertise and interest.*
- Humans are likely to be successful when interested, practiced, and well rested. Humans are more likely to be susceptible to fatigue. *Humans typically exhibit low accuracy/fidelity agents when subjected to stressful conditions, such as those mentioned in section 5.8.1, but unreliable software, hardware, or exposure of unpredictable environmental interference can also make mechanistic agents unreliable, if they fail to adapt in a situation that requires adaptation.*

In between these extremes, lies a spectrum of smart cooperative systems, that are essentially partnerships between humans and machines. The so-called Internet of Things is a scenario where this will become of prime importance.

From the comments above, we see that sudden *system change*, outside the limits of fault tolerances, whether planned or unplanned can bring about situations where agents perform inaccurately, leading to propagation of errors and faults, instead of the isolation and repair.

1.9.2 Accuracy of intended outcomes (passive assessment of anomalies)

Let's call error and faults 'anomalies', or undesirable situations within the context of a system. They are spanned by the two categories of system behaviour:

- Semantics (qualitative (human subjective) interpretation) - correctness
- Dynamics (quantitative (objective) measures) - focus or targeting

The role of promise theory is to make these two aspects of the system part of a common framework, hence promises can blur the distinction between them.

Semantics always operate within the constraints of dynamics (systems cannot be asked to do impossible things).

- **Semantic (qualitative) anomalies** (distortion or botching of an intended outcome)

  This is an anomaly that can be traced to a human interpretation, since it requires cognition, and/or a specification of intent. Semantic anomalies can happen:

  - By direct interaction of a human within the system. e.g. a wrong choice, based on analysis, perhaps because the system has become too hard to fully comprehend.
  - Through the programming of intent by proxy, e.g. in software, or a machine design-flaw, relative to its purpose.
  - Unintended emergent effects identified by interpretation.

  Promise theory separates promises to provide (+) and accept (-) from one another. Semantic anomalies can exist in both:

- (+) What is offered or executed is incorrect, ambiguous, misaligned with intent, or of the wrong type, etc
- (-) What is received is misinterpreted or misunderstood, or expects the wrong type.

Possible alleviations to semantic faults/errors include:

- We can try to keep systems simpler, within the realm of comprehensibility. Or we can try to take humans our of the system altogether, and remove the need to comprehend anything except a constraint model.
- The system is difficult to trust if it is non-linear, or unstable.
- Semantic averaging over multiple agents, e.g. pair programming, dual-agents for confirmation (redundant flight computers, human supervision (co-pilot), etc) by equivalent agents (e.g. coworkers of similar skill level).

- **Dynamic (quantitative) anomalies** (measurable parameters fall outside acceptable limits)

  A failure to meet performance expectations, perhaps during unusual circumstances. Failure to meet an SLA. Reach a capacity limit in hardware or software that throttles the behaviour, or leads to a crash. An agent misses a target and fails to achieve the desired goal.

1.10 Propagation, distortion, and loss of system semantics

One of the results of promise theory is a better understanding of what we can expect from intermediate agencies within systems. Because any person, device or agency, which propagates intentions and outcomes between two parties as a relay or intermediary, is completely free to distort those intentions and outcomes, agents have a fundamental blindness when acting through proxies. They have to trust the intermediaries without question.

1.10.1 Tampering by 'men in the middle' (serial distortion)

We can return to the details of this (known as the intermediate agent problem, or the end-to-end problem). Generally speaking, systems where outcomes are critical, should avoid proxies, or 'middle-men', as these are obvious places for distortion of intent and outcome. Each serial leg of dependency compounds the possible distortion.

Definition 17 (Tampering (serial distortion)) *Tampering is the imposition of a change on a system that is made contrary to a promised outcome.*

1.10.2 Crosstalk or channel separation (parallel distortion)

Agents that cannot discriminate between different contexts, in theit assessments, can introduce errors by inaccuracy, or misinterpretation of context. Such low fidelity assessments, i.e. errors of promise utilization, and miscomprehension, can lead to catastrophic changes of intent, especially where systems amplify rather than tolerate errors.

Definition 18 (Crosstalk (parallel distortion)) *The proximity of two agents, making different promises to a receiver, may result in a low-fidelity recipient agent failing to discriminate between the promises, hence resulting mixing the outcomes.*

Example 5 *Hitting the wrong button on a command console. Placing the intercom next to the ejector seat on an aircraft would be a poor design choice, since a small perturbation would have disastrous consequences.*

Alleviations are a general part of system fault-tolerance. They include greater clarity in the separation between contexts, according to the promised limits of perception for the agents who are the recipients. This might include visual and auditory channels for humans. Such considerations are a part of user interface design theory.

1.11 Assessment of outcomes

To make reliable assessments we need a coordinate basis, or a graduated axis along which to measure. Any agent can make an arbitrary assessment about any other's fidelity in keeping its promises, but we would like to do better. Assessments can be flawed: errors of measurement and interpretation are well known phenomena that draw attention to the fidelity of the assessing agents. With so many places for unintended distortion to take place, we begin to see the importance of tolerance and margins in system design.

1.11.1 Promises as a semantic 'coordinate basis' for measurement

No agent can assess another without a scale of reference, i.e. a measuring stick. If what was intended is not documented, then any outcome must be fair.

How shall we measure purpose? The measuring stick we use is not only quantitative, but is a choice, like a vector marking alternative directions. Promises play this role: they form a spanning basis for the intended outcomes of each agent's behaviours.

Every agent that in in scope of the promises has the ability to assess whether they were kept or not, relative to their context, either inside or outside the system limits. Outcome is relative to its promised purpose. Promises thus form a simple formalized basis against which to judge the state of the system.

Comment 7 (Functional assessment of systems) *Today it is common to add ad hoc monitoring of systems, with humans left as observers trying to divine meaning from the data. This can be useful as a process of scientific learning about the system, in order to gather motivation for policy, but it would be dangerous to act on such information without the prior motivation of a promise about what are considered acceptable limits on the measured values. Targeted testing is sometimes called acceptance testing or unit testing in software. This can be applied to any promise.*

Comment 8 (Ad hoc monitoring is unstable) *Without a semantic calibration, we cannot know whether assessment is significant or not. Measuring or monitoring systems, without any promise of what to expect, leads to speculation about the significance of measurments. This can lead to divergence of opinion and unstable actions. Acting on an ad hoc assessment, without a promise about its expected behaviour within a larger model, could easily be disastrous, according to another agent's assessment.*

1.11.2 Foundation of assessement: relativity and basis scale

There are several parameters to consider in assessing outcomes. In any interaction, the transfer of intent depends on all the interacting parties:

- *Semantic overlap*: does the sender make a promise that is of a compatible type to that the receiver depends on? Is the type of the promise bodies b_+ and b_- the same? For example: if an agent accepts apples, providing oranges would not be acceptable.

- *Dynamic overlap*: does the receiver accept that the magnitude of the promise given by the sender satisfies its expectations? For example, an agent promising 6 kgs of apples would not be enough to satisfy an agent promising to consume $8 - 10$ kgs.

  Dynamical measurements often relate to space and time.

  Temporal overlap: is the promise kept within an acceptable time frame? How often should we sample the promiser to know whether the promise is kept. A promise that is too slow is the same as a promise not kept at all. e.g. promise to deliver same day, but delivery takes 2 days.

  Spatial overlap: does the right agent, in the right location, make the right promise?

1.11.3 Drift into failure

In his book Drift into Failure[Dek11], S. Dekker describes how 'failure' often creeps up on the unwitting users or participants. As systems are used, and maintained, they are improved and adapated. Gradual adaptations, changing the envelope of apparently innocent parts have unexpected consequences as the changes are amplified.

Flawed expectations, lack of knowlege or awareness of the actual behaviour of the 'unruly system' all contribute to a gradual drift of a system out of its fitness envelope and into the realm of unreliability. Problems with change (errors), lead to problems of state (faults), and expose problems of inadequate design (flaws). In other words, failures occur because systems are not simply not static. They undergo planned and unplanned change, driven by environmental forces we often don't even realize we need to know about[HWL06, Bur13].

Could we begin to address the build up of drift? This seems plausible, in a limited sense. Managing drift is what maintenance routines are for, and tools can help. Expectation creep can be stopped by mechanical oversight, with independent auditing[8]. The key is to have a declaration of intended outcomes (promises) that can be monitored in a targeted fashion, without pressure from other concerns to relax standards. The trouble is that we can't even define drift unless we know what the system is meant to promise. Even then, the promises might be naive. This is the challenge. I will return to this in the chapter on scaling.

1.12 How to use the tools described in these notes

The material in these notes presents some tools to help express, measure, and reason about systems, and the properties that lead to resilience. It is not a recipe for 'success'. It may or may not tell us directly how to improve the resilience of a system. It certainly points to some simple principles and strategies. Nevertheless, it's important to realize that resilience is an arms race. It's not what you know that helps you the most, but often what you don't know that that points to new considerations.

Designing a system, which explicitly promises and documents intent, allows us to gradually test hypotheses, in real time, and with focused measurement. Generic monitoring is useless. You might eventually be able to infer whether a person is doing a good job by measuring their pulse and core temperature, but you will learn the answer much more quickly by measuring whether they keep their promises. Ultimately though, if you don't know what to measure and promise about a system, then you need a leap of imagination or a bad experience, rather than a leap of algebra, to help you. This is the nature of mathematics and of systems.

Use the tools here to sharpen your understanding of the underlying mechanisms, to sharpen your descriptions of systems. Develop your expertise to make and keep promises by interacting and learning directly from multiple vantage points throughout the system.

[8] The issue of independent oversight was how Promise Theory came about, from CFEngine's model of system maintenance[Bur95, Bur04b].

Chapter 2

Classical reliability theory and the limit of perfect cooperation

In the traditional approach to reliability, a system is considered to be a collection of components with some probability of failure. So-called *structure functions* are used to replace detailed functional semantics with a bare minimum model of dependencies between black box components. Components are assumed either to be working or not working. Reliability is then quantified by an imagined probability that components are independently in a working state.

In this chapter, we look at the basic construction of the classical method, and show how it maps into promise theory, in a simplifying limit. Since we aim to go beyond the classical results, we need to see how its generalization reduces to the same result, with the same assumptions.

2.1 The aims of classical reliability

The classical approach to system modelling was constructed out of statistical theory, with a particular aim in mind. The idea was not that component probabilities could be assessed individually for specific systems, but rather that they could all be modelled by common results from generalized Poisson statistics. It is known from manufacturing processes, that component lifetimes often follow a Poisson distribution, and hence this can be used to make generic predictions about likely time before a failure occurred.

Classical reliabilty theory is a blunt tool for making broad estimates: all details are lumped together into a single value, represented by the structure function. Strategies for increasing the probability of being in a working state could be addressed by reducing dependency, or increasing redundancy of the components.

2.2 The assumptions

The reliability of systems of components is traditionally studied through the so-called *structure function*[Nat98, HR94]

$$\phi(x) : x \to \{0, 1\} \tag{2.1}$$

which expresses the variable dependence of components on one another, and an expectation of being in a working state (basically a frequency probability or reliability). When faults occur, its value is reduced, as a component variable goes from 1 to 0. It is computed following the basic laws of probability in series and parallel. Components are independent, and their cooperation is implicitly encoded by the structure function's form. The structure function approach is characterized by:

- Maximal impartiality and suppressed semantics.
- Being generic and unspecific in its assessements.

As a consistency check on this extreme end of the spectrum of interpretation, we shall demonstrate, in this chapter, how a promise theory view reproduces the main results of the classical theory, before going beyond them. This helps to show where the limitations lie.

Comment 9 (How shall we interpret probabilities in reliability theory?) *Probabilities are used in different ways. Here we are using them as estimators for the fraction of samples over which a system is in a working state, i.e. as an approximate scale factor for reliability. The values are not computed or measured, or even guessed normally. One is more interested in how combinations of the probabilities are larger or smaller for the sake understanding how the likelihood of being in a working state is affected by combinatorics.*

2.3 Quantitative reliability - traditional approach

In the following sections, we repeat some of the standard results about component analysis from reliability theory.

Given how these diagrams resemble flow diagrams for electrical components, it is natural to think of this as implying flow, but this is somewhat misleading. For the serialization, there is a transmission of service from left to right in each image, only insofar as what happens at the right hand side implicitly depends on what happens on the left hand side. However, the components' faults are treated as independent, so the failure of one component does not necessarily transfer additional load to another. They allow for that kind of effect, we need to study the examples more closely. Promise Theory is helpful here because it documents these relationships atomically.

2.3.1 Conditional promise law (dependency)

To compute dependencies, we need to recall the law of conditional promises. This law defines what it means for an agent to treat promises made conditionally in a fashion consistent with the realities of what the agent can be responsible for. An agent may only make a promise about its own behaviour.

Dependency enables delegation and specialization, but it also brings potential fragility to any cooperative system. Delegation is a strategy that can be used well or poorly. It is modelled with conditional promises.

The conditional promise law says that a promise, conditional on a dependency, cannot be assessed as a true promise unless the dependency is promised and accepted by the promiser of the conditional promise. Thus, the promise to deliver a package by FedEx:

$$\text{Shop} \xrightarrow{+\text{deliver}|\text{FedEx}} \text{Customer} \tag{2.2}$$

cannot be assessed by the customer without the full construction:

$$\text{Shop} \xrightarrow{+\text{deliver}|\text{FedExService}} \text{Customer} \tag{2.3}$$

$$\text{Shop} \xrightarrow{-\text{FedExService}} \text{Customer} \tag{2.4}$$

$$\text{Shop} \xrightarrow{-\text{FedExService}} \text{FedEx} \tag{2.5}$$

$$\text{FedEx} \xrightarrow{+\text{FedExService}} \text{Shop} \tag{2.6}$$

$$\tag{2.7}$$

i.e. the Shop must not only promise that it will deliver a package, but that it will accept the services of the FedEx agent in order to do so. This construction forms the basis of conditional chaining of components in systems.

2.3.2 Serial dependency of components

In the component structure view, the diagrams are symmetrical and reversible, even when there is an implicit arrow of flow. in a promise formulation, the asymmetry of causal direction is clear. In the serial case (see fig 2.1), agents A AND B have to work, and the probability

$$P(A \textbf{ AND } B) = P(A)P(B|A). \tag{2.8}$$

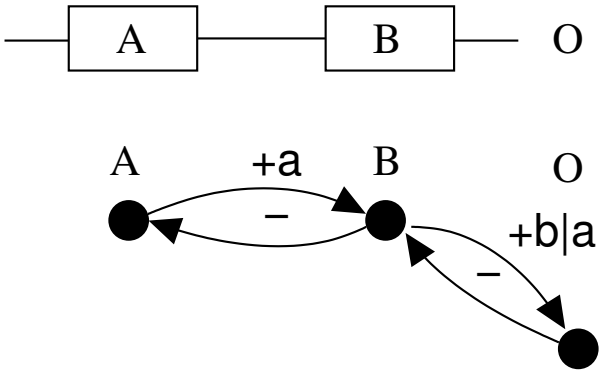

Figure 2.1: Serially dependent components and promises. Note how the causal flow direction is encoded into the promise version but is only assumed in the structure function.

If A and B are independent, this is simply $P(B|A) = P(A)P(B)$. Or in terms of the structure function, using the convention $P(A) = p_A$, etc, we have

$$\phi_{AB} = \phi_A \phi_B \tag{2.9}$$

$$E_{\rm all}(\phi_{AB}) = p_A p_B, \tag{2.10}$$

i.e. the expectation of all parties that this arrangement is in a working state is the product of probabilities ot the components being in a working state. The general rule for combination becomes:

$$E_{\rm all}(\phi_{\rm tot}) = \prod_i p_i \tag{2.11}$$

In terms of promises, we have:

$$A \xrightarrow{+a} B \tag{2.12}$$

$$B \xrightarrow{-a} A \tag{2.13}$$

$$B \xrightarrow{+b|a} O \tag{2.14}$$

i.e. A promises $+a$ to B, B promises to use that, and furthermore B promises conditionally b given that is has received a promise of a. And the end-of-chain observer's O's expectation that the promise from B is kept is:

$$E_O(B \xrightarrow{+b|a} O) = P(b|a) = p_B p_A, \tag{2.15}$$

giving an alternative derivation.

2.3.3 Redundant components—alternative handlers

For alternative components (see fig. 2.2), there is redundancy, e.g. pilot and co-pilot, master and backup, etc. Although they are positioned in parallel, they do not necessarily complement one another in terms of performance throughput[1]. Either or both of the components should work, so the reliability frequency 'probabilities' combine:

$$P(A \textbf{ OR } B) = P(A) + P(B) - P(A \textbf{ AND } B) \tag{2.16}$$

The last term only applies if both components are in play, in which case there is some overcounting where the frequencies overlap. Alternatively:

$$\begin{aligned} P(A \textbf{ OR } B) &= \textbf{ NOT } ((\textbf{ NOT } P(A)) \textbf{ AND } (\textbf{ NOT } (P(B))) && (2.17)\\ &= = (1 - (1 - p_A)(1 - p_B)) && (2.18)\\ &= = p_A + p_B - p_A p_B && (2.19)\\ &\equiv p_A \coprod p_B && (2.20)\\ &\equiv P(A) \coprod P(B). && (2.21) \end{aligned}$$

[1] One of the weaknesses of this simple model is that this is unclear, but we return to discuss this issue in the next chapter

The exclusive expectation that only one and not the other is working is:

$$P(A \ \mathbf{XOR} \ B) = P(A) + P(B) - 2P(A \ \mathbf{AND} \ B), \tag{2.22}$$

i.e. excluding the overlap region altogether. The promise version of this (see figure) has

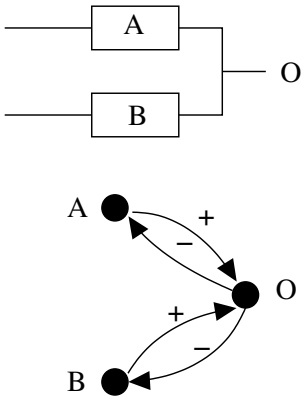

Figure 2.2: Parallel components / promises

$$A \xrightarrow{+a} O \tag{2.23}$$
$$B \xrightarrow{+b} O \tag{2.24}$$
$$O \xrightarrow{-a} A \tag{2.25}$$
$$O \xrightarrow{-b} B \tag{2.26}$$

The observer's estimate of the reliabilities for promise keeping is:

$$E_O\left(A \xrightarrow{+a} O\right) = p_A \tag{2.27}$$
$$E_O\left(B \xrightarrow{+b} O\right) = p_B \tag{2.28}$$

By treating A and B as a coarse super-agent, we may write:

$$O \xrightarrow{-(a \ \mathbf{OR} \ b)} \{A, B\} \tag{2.29}$$

whence , the expectation of being able to keep the promise

$$E_O(\{A, B\} \xrightarrow{+(p_A \ \mathbf{OR} \ p_B)} O) = p_A + p_B - p_A p_B, \tag{2.30}$$
$$= p_A \coprod p_B. \tag{2.31}$$

and we recover the result in a different way.

2.4 Combining dependency and redundancy (serial and parallel)

The structure diagrams, with their combination of serial and parallel arrangements tempts us to think about electrical circuits and continuous flow, but this may be misleading. Systems may involve discrete events or continuous operation.

2.4.1 Redundant arrangement of dependent serial sub-systems

Duplicating whole systems in parallel is a form of redundancy at the high level, e.g. a server and a backup server, or a route and a backup route. Each system may consist of sequences of components, like the dotted boxes in figure 2.3.

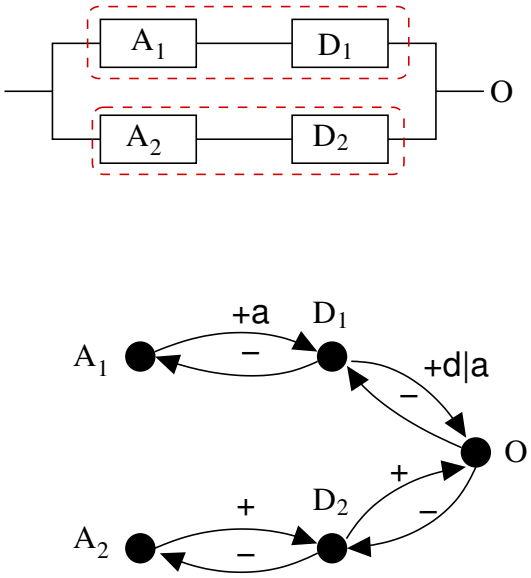

Figure 2.3: Redundant 2-stage serial chains (dotted) in parallel

Taking the simplest example of a system of two serial agents A_1 and dependent series component D_1, in parallel, with a complete clone A_2, D_2, the structure functions or reliabilities have the form:

$$\phi = \phi((A_1 \textbf{ AND } D_1) \textbf{ OR } (A_2 \textbf{ AND } D_2)) \quad = \quad \phi(A_1)\phi(D_1) \coprod \phi(A_2)\phi(D_2) \tag{2.32}$$

Defining the reliability expectations:

$$\begin{aligned} \phi(A_1) &= P(A_1) = x_1 && (2.33)\\ \phi(D_1) &= P(D_1) = x_2 && (2.34)\\ \phi(A_2) &= P(A_2) = y_1 && (2.35)\\ \phi(D_2) &= P(D_2) = y_2 && (2.36) \end{aligned}$$

Here, by analogy with the cases above, the $P(A)$ etc, may be viewed as the probability or reliability fraction with which the agent A keeps its promise. We only need to describe precisely what (and to whom) that promise is. The arrangement of components in the structure diagram implies to whom the promise is made, and on whom it depends, but it does not matter too much what promise is being kept. This is built into the assumptions of such classical reliability treatments. With that, we can simply say that the total structure function is:

$$\begin{aligned} \phi &= x_1 x_2 \coprod y_1 y_2 && (2.37)\\ &= x_1 x_2 + y_1 y_2 - x_1 y_1 x_2 y_2 && (2.38) \end{aligned}$$

Also, we can supplement the promise sematics to complete the functional description that is absent in the structure viewpoint:

$$\begin{aligned} A_1 &\xrightarrow{+a_1} D_1 && (2.39)\\ D_1 &\xrightarrow{-a_1} A_1 && (2.40)\\ D_1 &\xrightarrow{+d_1|a_1} O && (2.41)\\ O &\xrightarrow{-d_1|a_1} D_1 && (2.42) \end{aligned}$$

Similarly, an analogous set of promises is true for the parallel system A_2, D_2. The two systems come together only at the choke point of the observer:

$$\begin{aligned} O &\xrightarrow{-d_1|a_1} D_1 && (2.43)\\ O &\xrightarrow{-d_2|a_2} D_2 && (2.44) \end{aligned}$$

From which O interprets, by coarse graining:

$$O \quad \xrightarrow{-d_1|a_1 \ \mathbf{OR} \ -d2|a_2} \quad \{D_1, D_2\}. \tag{2.45}$$

Note how the implicit assumption of aggregation and coarse graining away differences between the branches is documented explicitly in the promise version of this formula. From this, it would infer the reliability of the system to be

$$\begin{aligned} E_{=}\left(\{D_1, D_2\} \xrightarrow{+d_1|a_1 \ \mathbf{OR} \ +d2|a_2} O\right) &= P(d_1|a_1) \coprod P(d_2|a_2) & (2.46) \\ &= x_1 x_2 \coprod y_1 y_2. & (2.47) \end{aligned}$$

This reproduces the result using the promise theory. Once again, the promise viewpoint doesn't alter the structure of the problem. It only adds the functional documentation of intent. This becomes more significant as the promise collaboration graphs become more complicated.

Comment: it is easy to see why there is still room for improvement in this design: the point of system recombination is itself a possble point of failure, with everything riding on a single consumer O.

2.4.2 A single system made from fully parallelized (redundant) components

The natural extension to removing a single point of failure is to make each functional component in the system independently redundant (see figure 2.4). This keeps all the failure probabilities independent and one expects the result to be smaller. Indeed, there is a well known theorem that proves this to be the case (see section 5.7.4). From

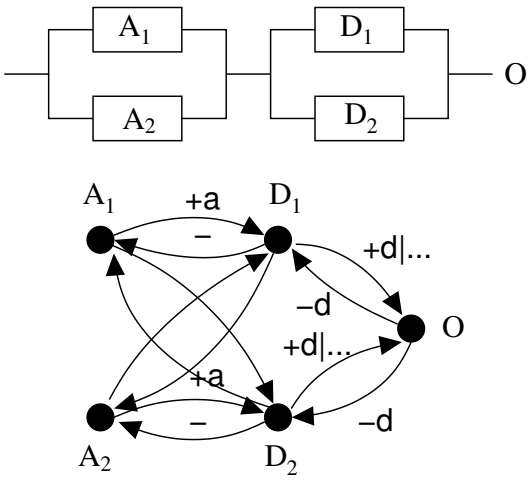

Figure 2.4: Serial chain of parallel components

the structure function, we now have:

$$\phi = (\phi(A_1) \ \mathbf{OR} \ \phi(A_2)) \ \mathbf{AND} \ (\phi(D_1) \ \mathbf{OR} \ \phi(D_2)) \tag{2.48}$$

Once again. defining the reliability expectations:

$$\begin{aligned} \phi(A_1) &= P(A_1) = x_1 & (2.49) \\ \phi(D_1) &= P(D_1) = x_2 & (2.50) \\ \phi(A_2) &= P(A_2) = y_1 & (2.51) \\ \phi(D_2) &= P(D_2) = y_2 & (2.52) \end{aligned}$$

This becomes simply;

$$\phi = \left(x_1 \coprod y_1\right)\left(x_2 \coprod y_2\right) \tag{2.53}$$

Now let's do it from the promise perspective. The promise diagram takes the form of a coupled set of agents. From the two row of diagram:

$$A_1 \xrightarrow{+a_1} D_1 \tag{2.54}$$
$$D_1 \xrightarrow{-a_1} A_1 \tag{2.55}$$
$$A_1 \xrightarrow{+a_2} D_2 \tag{2.56}$$
$$D_2 \xrightarrow{-a_2} A_1 \tag{2.57}$$

and the bottom row:

$$A_2 \xrightarrow{+a_2} D_2 \tag{2.58}$$
$$D_2 \xrightarrow{-a_2} A_2 \tag{2.59}$$
$$A_2 \xrightarrow{+a_1} D_1 \tag{2.60}$$
$$D_1 \xrightarrow{-a_1} A_2 \tag{2.61}$$

Now there are two stages of OR aggregation: each of the D_i agents, and finally the observer O. The D_i aggregations take the form:

$$D_1 \xrightarrow{-(a_1 \ \mathbf{OR} \ a_2)} \{A_1, A_2\} \tag{2.62}$$
$$D_2 \xrightarrow{-(a_1 \ \mathbf{OR} \ a_2)} \{A_1, A_2\} \tag{2.63}$$

Thus, D_i promise to O:

$$D_1 \xrightarrow{+d_1|(a_1 \ \mathbf{OR} \ a_2)} O \tag{2.64}$$
$$D_2 \xrightarrow{+d_2|(a_1 \ \mathbf{OR} \ a_2)} O \tag{2.65}$$

By further aggregation, this is observed as:

$$O \xrightarrow{(-d_1|(a_1 \ \mathbf{OR} \ a_2)) \ \mathbf{OR} \ (-d_2|(a_1 \ \mathbf{OR} \ a_2))} \{D_1, D_2\} \tag{2.66}$$

Hence, O evaluates the reliability of this promise, with the coarse graining assumption made explicit once again:

$$E_O\left(\{D_1, D_2\} \xrightarrow{(+d_1|(a_1 \ \mathbf{OR} \ a_2)) \ \mathbf{OR} \ (+d_2|(a_1 \ \mathbf{OR} \ a_2))} O\right) = \left(x_2(x_1 \coprod y_1)\right) \coprod \left(y_2(x_1 \coprod y_1)\right) \tag{2.67}$$
$$= (x_1 \coprod y_1)(x_2 \coprod y_2) \tag{2.68}$$

which reproduces the earlier result.

2.5 The folk theorem for redundant fault tolerance

The redundancy folk theorem[Nat98, Bur04a], of classical reliabilty theory, states that a single system in which every component is duplicated for redundancy is at least as reliable as duplicated systems offered in parallel (see figure 2.5). This follows from the lemma:

$$\phi(x \coprod y) \geq \phi(x) \coprod \phi(y) \tag{2.69}$$

for any $\phi(\cdot)$. When applied recursively, this purely mathematical inequality implies that low level redundancy is the most reliable approach in terms of fault coverage. The assumptions of this are based entirely on probabilities however.

Although the probabilities satisfy the inequality without prejudice, a single system with every component made reundant might not be the cheapest approach to build. For example, we might not know how to make the system work with that design. Alternatively, the system represented by $\phi(x)$ might be a cheap commodity, making an array of cloned systems cheaper than a single system of higher quality.

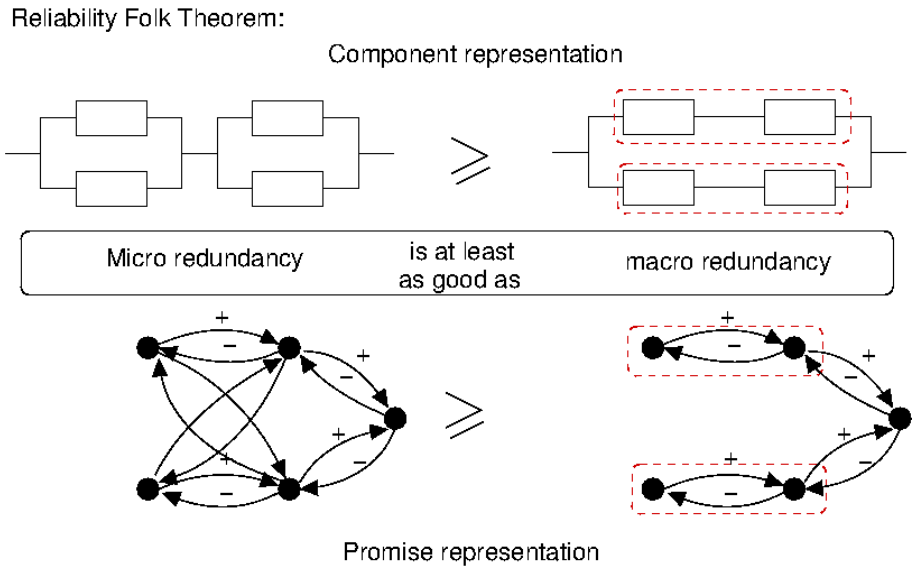

Figure 2.5: Illustration of the redundancy folk theorem.

By examining the promise graph for low level redundancy, in figure 2.4, we see that the number of promise required to make all necessary interactions complete, exposing all hidden assumptions, is twice the number for the low level redundancy case, thus the chance of a system keeping all of the design promises is greatly reduced. This works against the 'truth' of the theorem.

In spite of the shortcomings, we can now write the folk-theorem result in a promise notation. For want of a better shorthand, and out of respect for tradition, we could express the promise lists in the simplest analogous form as:

$$E\left(\pi(\langle A_1, O, b_1\rangle \coprod \langle A_2, O, b_2\rangle)\right) \geq E\left(\pi(\langle A_1, O, b_1\rangle) \coprod \pi(\langle A_2, O, b_2\rangle)\right). \tag{2.70}$$

This shows that it is purely as result about structural expectation. We could add to this a cost function $C(\cdot)$, which has the opposite result:

$$C\left(\pi(\langle A_1, O, b_1\rangle \coprod \langle A_2, O, b_2\rangle)\right) \leq C\left(\pi(\langle A_1, O, b_1\rangle) \coprod \pi(\langle A_2, O, b_2\rangle)\right). \tag{2.71}$$

The proof is trivial by counting interaction lines and associating a non-zero cost to each promise. We can add to this the cost of repairing components, or replacing components, which is the same in both cases, unless the components themselves are aggregated into commodity blocks for cost savings.

The lesson from this exercise is that, in any system of significant intricacy, there are no simple rules for optimizing reliability. The statistical theorems are interesting as guidance, but they do not offer a panacea for design choices.

2.6 Fault trees

A natural extension of the component approach to reliability is Fault Tree Analysis (FTA), widely used in the nuclear power industry. It suffers from the same limitations as the examples above, but allows for more sophisticated structures. It takes a simple hierarchical view of system dependency, that is quasi-deterministic, and hence assumes sufficient isolation from other effects.

The main purpose of FTA is to enable computation of relative reliability. It fails to model semantics in a way that is related to the structure of the system. To address these concerns, we need to go to a graphical method: promise theory.

2.7 Queues

Queues are the balance between arrival processes and service or renewal processes. If we imagine the arrival process is the arrival of faults, then we can measure system rejuvenation statistically using simple flow rates. This was approach was used to discuss scaling in [BC04]. This approach can give some indication about timescales and fault tolerance limits, using the queueing instability[Bur04a].

2.8 What is the limit of perfect cooperation?

The promise representation has the ability to represent more aspects of a cooperative system of component parts than the structure function approach. However, both give the same result in the limit in which promise relationships are mutually established with 100 percent certaintly. The limit implies that:

1. Both sides agree to cooperate.
2. Cooperation implies propagation of influence with unit probability.

We can now work on generalizations that stray from this limit. In general, this means that the likelihood of a working system will be estimated lower than the classical limit, but the important part is not the numbers (which are only estimates), but rather how the scaling of probablity is affected by different issues with greater specificity.

2.9 Why not logic and rules?

In Western culture, we have a strong belief in rules, so much so that it pervades our thinking. We stop at red lights, and walk at the green man. We speak of 'laws of nature' as if the world were following a rigid plan; but, when we say 'exception to the rule', we really mean 'exception to the norm'. Do we stop at stop lights because we have to, or because it's a good idea to comply with conventions?

The problem with this trust in rules and obligations is that they distort our understanding of reality. Systems do not have to comply with our expectations. It's the other way around: we try to build systems that are stable enough to allow us to make rules by observing their normal behaviour. But then, we come to believe that what is normal, in one context, is actually a generally applicable rule, or even 'truth'. This is where flaws begin. Imposing rules and logical assertions onto systems is an interesting exercise for the purpose of testing, but it does little to promote understanding. Turning this upside down, a promise viewpoint replaces rule and obligation with likelihood and promise, to describe our design understanding as best effort.

2.10 Summary

In this chapter, we've used the classical approach of treating components as black boxes. They are assumed to work (or not) with a certain probability. We do not model *how* they work together, only that they are connected through dependencies and with redundant alternatives. This offers some rough statistical insights about the ability to repair parts within the design.

Although we need approximate methods for summing up statistical reliabilities at fixed semantics, we also need to be able to ask deeper questions related to functional design and changing semantics. What if the semantics of a system suddenly changed, while it continued to function? That would also be a fault, in which semantics no longer represented intent. These are the issues that relate to human involvement in systems, and thus represent a crucial perspective, from the design of machines as proxies for our intentions, to our own interactions with them.

Chapter 3

Agents, promises, and interactions

In this chapter, we attempt to ask: could we predict certain failures of a system, based on a deeper understanding of the semantics and dynamics of its operation? I will argue that this is plausible using promise theory, because:

- It leads to a directed graph of intent, which allows many standard techniques for the analysis of outcomes.
- It encodes more information about how a system works than a component model.
- It can encode and take into account *context*, which permits a fine graining of causal pathways, based on semantics.

The agents of promise theory will henceforth be the atomic building blocks, from which a greater chemistry of cooperation and functional design emerges. We may use agents to model all kinds of components, whether 'dumb' or 'smart', human or machine. They are distinguished only by the promises they make.

3.1 Shortcomings of classical reliability theory

Classical reliability theory was addressed to the desciption of machine components in which the fixed meanings of components were assumed to be correctly composed to deliver a workable outcome, for simple networks. The only question remaining was: does any component stop playing its part? Consequently, it was never intended to encompass human issues, qualities, or to represent changing systems.

In any statistical theory, the assumptions and semantics are constant in order to have parity when measuring data. The functional behaviour of a system must therefore be constant too, unambiguous and inevitable. The problem with this view is that functional utility only makes sense relative to certain contexts or circumstances. In modern systems, humans and machines interact in integrated systems and a purely statistical theory is insufficient to describe system reliability. This suggests, too, that probability as an estimator of likelihood is not the right interpretation, but probability as a scale factor for relative weight in a cooperation is.

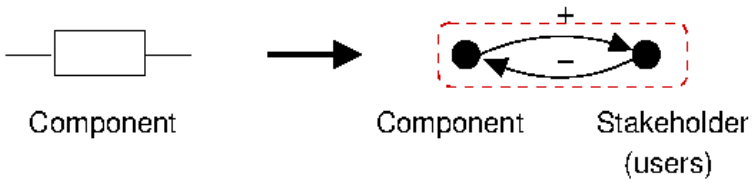

Figure 3.1: The shift from traditional statistical reliability theory to a promise view. Promises represent a shift attention from standalone components to interactions with users.

Let's qualify some of the shortcomings of the traditional approach. Using the structure function to represent the system behaviour has limited expressive power:

1. **It focuses on components as lumped 'roles' instead of the integrity and appropriateness of their interactions**. No component does anything by itself. Thus even if all components are working, if they fail to

interact properly, the system will not work. Every interaction with uni-directional propagation of influence involves two promises; one to provide and one to accept. Thus, any dependence can fail in one of two different ways:

- By sender/provider/source promise not kept (+).
- By receive/accepter/sink promise not kept (-).

All transmission of intent, every act of cooperation, involves two independent agencies. The two parts are independent promises, located at independent agencies, but they form dependent failure modes: the receiver cannot receive if the sender has not sent, so it is dependent. This is not accounted for in the basic fault theory, where it is assumed that systems work by imposition / obligation.

2. **Perfect cooperation without situational awareness**. Because it lumps everything into single probabilities, combined determinisitically, it effectively assumes that cooperation and adaptation of agents plays no role in its effective functioning. The agents play their roles without question or self-awareness. This could be appropriate for 'dumb agents', such as manufactured components, whose behaviour does not depend on their perception or awareness of their situation, but it is not adequate for human-machine collaborations.

3. **Limits and tolerances**. The structure function does not easily account for what happens when the sum of impositions on a component exceeds the component's capacity to perform. Capacity limits are not included in the simple expressions for structure, but they are a key part of information theory and transmission of influence.

   When a limit is exceeded, one of two possibilities must take place:

   - A queue must build up at sender or receiver.
   - Certain impositions or promises to use must be abandoned.

4. **Scale anomalies**. As workload is scaled to higher levels, we want to understand how the fixed scale limitations of individual components impacts the total system. Bottlenecks could be mitigated by employing parallel processing, not merely redundancy, but this causes a potential confusion between parallelism and fault redundancy, which serve different purposes.

Fortunately, the results of the previous chapter can easily be improved on by developing the promise approach, while keeping the simplicity of the method.

3.2 Promises and their relationship to faults

A promise is an intention that is documented as a tuple of components.

$$\pi \quad : \quad \langle S, R, b(\tau, \chi_\tau) \rangle, \tag{3.1}$$

where S is a sender of a promise, R is a receiver of a promise, and b is the promise body, which has type τ and constraint χ_τ. Promises are not directly observable, but their outcomes can be assessed, relative to the states of a given agent.

$$A(\pi) \quad : \quad \langle S, R, b(\tau, \chi_\tau) \rangle \to 0, 1 \tag{3.2}$$

$$E(\pi) \quad : \quad \langle S, R, b(\tau, \chi_\tau) \rangle \to [0, 1] \tag{3.3}$$

where 0 corresponds to 'not kept' and 1 corresponds to kept. The usual interpretation of a fault is the observation that an assessment, made by some agent, $A(\pi) = 0$, or that the average assessment over time $E(\pi)$ falls below some policy threshold. But how is this mapping made? What factors affect the assessment of a promise?

While physics and statistics focus primarily on describing change (dynamics)of entities (particles, atoms, etc), promise theory attempts to unify three essential parts of agents within a system: dynamics, semantics and context (see fig. 3.2). These aspects form complementary views of system descriptions (see table 3.1)

The table and figure 3.2 lay out the essential aspects of systems along three main axes, and their combinatorics. Promise-oriented reliability extends the expressivity and ambitions of the statistical component models in several ways:

ASPECT	COLLOQUIAL	REALIZATIONS
SEMANTICS	MEANING	ASSUMPTION, INTERPRETATION, MEANING, QUALITY, GRAPHS, PATHWAYS, STRUCTURE, CONSTRAINTS
CONTEXT	SELECTION	STATE, AWARENESS, CONDITIONS, ENVIRONMENT, MEASUREMENT, PHASE-SPACE
DYNAMICS	CHANGE	QUANTITY, CHANGE, TRANSITIONS SCALARS, VECTORS, PROBABILITIES, SCALE

TABLE 3.1: THE THREE ASPECTS OF AGENCY.

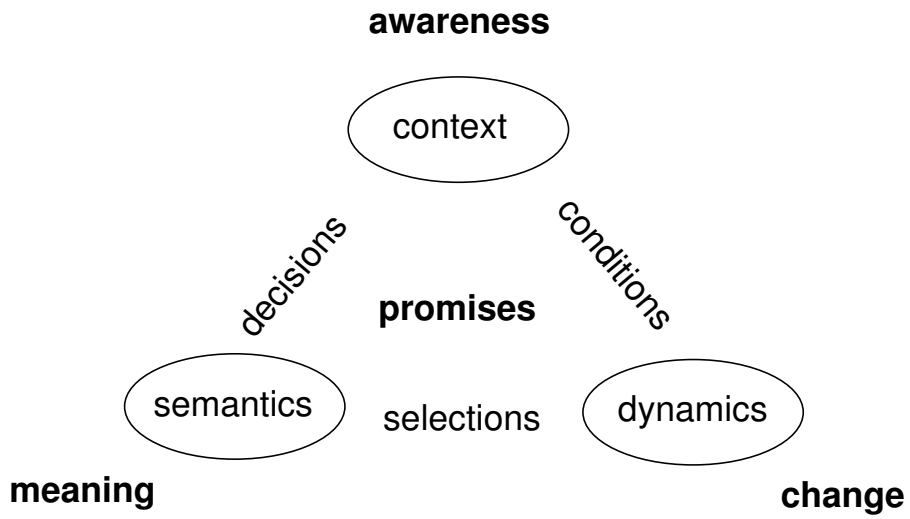

Figure 3.2: The three aspects of systems for a unified treatment. In between context and interpretation, lies decision-making, to guide system dynamics. Between context and dynamics, we find the state or condition of the environment, to which we attribute semantics. We attach meaning to change by selecting favoured outcomes, in a given context. Finally, we attribute meaning to current state and desired change by making promises about desired outcomes, i.e. we document what we intend.

- It shifts the attention to successful interactions between agents, rather than indepedent working state of agents. Hence it represents and acknowledges delocalization of responsibility.
- It allows us to model smarter agents that have memory and can adapt to different contexts.
- It quantifies even qualitative aspects of system behaviour, adding *dimension* or *measure* to the assessment of working state: how well, how fast, how according to intent (specification)?

Definition 10 describes a fault as a condition in which the actual state of the system does not match the promised (desired) state. In many cases, we depend not only on single promises, but promise bindings ($\pm$ neutrality) for cooperation. In other words, instead of thinking 'component is working', we think: is the component playing its role to all the stakeholders who rely on it? This does not only mean those components that are directly connected to it, but also those, which might be far removed from it, and still rely on its behaviours, unlike in the classical component *in situ* model. The promise arrows are virtual relationships, not physical connections.

3.3 The basic promise failure modes

Having identified the ways to define faults and errors consistently in terms of promises, we can consider how the failure to keep certain promises can lead to these conditions.

3.3.1 Standalone agent promises

Promises to self, to no one in particular, or to purely passive stakeholders who do not act or depend explicitly on a promise, represent the simplest declarations of intent (see fig. 3.3). Such promises may assessed to be either be kept or not kept, by any agent in scope, i.e. who knows about the promise.

Each promise acts as a local 'measuring scale' for assessing system characteristics along different 'axes', like in a coordinate system.

Example 6 *If a person promises to tie their shoe laces, brush their teeth and wash behind their ears, then we have three axes on which to measure and compare the state of person.*

Each promise type becomes an axis for measurement: a possible degree of freedom for agent behaviours. Assessments from kept to not kept (either on a binary or on a continuous scale), allow us to collect data meaningfully about whether a system is measuring up to its intented goals. Without any promise, we cannot measure the success or failure of a system, we can only study patterns of behaviour[1].

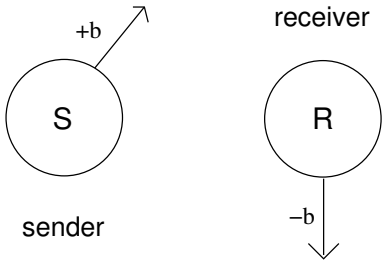

Figure 3.3: Single agents may make promises to give (+) or accept (-) something from another agen. Note that it is the sign of the promise that denotes direction of flow, while the arrow is always the direction of intent, originating from the responsible agent

The simplest reliability question we can ask of a system is thus:

- Was any promise made relating to observed behaviour?
- Was the promise kept?

Answering these questions help to determine whether

- The system is behaving as designed, but it does not currently align with its stated goals (design flaw).
- The system is behaving as designed, but an observer's can't or won't accept the behaviour (inconsistent or mismatched promises).
- The system is behaving contrary to design (simple assessment of fault).

Promises act as calibrators for fault detection. In the traditional black box model, all of these modes of failure (above) are random faults, and the cause of the fault is unknown. We cannot resolve in which specific agency the misalignment with intention lies. But, by making explicit promises that document both qualitative and quantitative constraints, we introduce a measuring stick for the design of a cooperative system.

Example 7 (Security breach) *A condition that is not easily defined as fault in the classical theory of reliability is a security breach. In a promise system we are able to define what specific promised condition was violated to lead to a security breach. In this way, security just becomes a promise, and a breach is only a fault or a particular kind of promise not kept.*

[1] Science is about measurement, and it is usually unprejudiced about the behaviours of its subjects. In the case of technology, the subjects are designed deliberately to have semantics; hence that becomes part of their behaviour. A promise creates an abstract measuring scale for semantically charged, purposeful outcomes.

Comment 10 (Monitoring and alarms) *Promise keeping also applies to all forms of measurement and monitoring for fault detection. Measuring without a stated promise is a waste of time, unless you are a researcher or investigator in search of clues, because the monitor doesn't know what to look for. In most IT monitoring systems, if any promises are made at all, they are usually ad hoc time series thresholds that are set by gut feeling, not aligned with system design promises.*

3.3.2 Timing of promise keeping and assessment (Nyquist sampling frequency)

A promiser that tries to keep its promise too late has not kept its promise, and the promisee interprets this simply as a fault. Responding within a timely fashion might be considered part of an extended protocol or handshake.

Using information theory, we can apply Nyquist's sampling theorem, in reverse, to infer that the information propagation density needs to be at least twice that of the dependency sampling, in other words: in order to keep a continuous, time-varying promise, when the promisee uses the promise at a frequency ν, the promiser will need to ensure its accuracy at the Nyquist frequency of 2ν in order to track the changes.

Law 2 (Nyquist frequency for promise maintenance) *A dependency should maintain or update its current state at (at least) twice the rate at which the promise changes are used as a dependency to avoid a perceived fault.*

System with periodic sampling can optimize this maintenance cycle for rapid repair. This changes the relative economics of redundanct, replacement, versus repair, discussed in chapter 5.

3.3.3 Coverage: behaviours that are promised and not-promised

Systems exhibit two kinds of behaviours:

- Intended or promised behaviours (what we normally call agency)
- Unintended behaviours (disrupted or emergent behaviours, from environmental noise and interruptions), which are *ad hoc* as there is no promise associated with them.

So a system is either guided or not. If it is not guided, there is not much we can say about its behaviour except by watching and trying to learn. If a system that does not make promises exhibits apparently predictable behaviour, we might infer that it does in fact make a concealed promise. Sometimes we believe in promises that have not been made, and speak about emergent behaviours.

Emergent behaviours are observed behaviours that resemble intentional behaviours, and may recognized and named, but where no promise was actually made[BB14]. These are typically collective behaviours, but taking into account scaled super agents, we might not be able to tell the origin of emergence.

Definition 19 (Emergent behaviour) *Observed behaviour, which is consistent with the existence of a promise, but has not been promised explicitly.*

In promise theory, we represent all behaviours by transition *trajectories* through the possible *states* available to agents[2]. These trajectories may be largely explained to be the correlated outcomes of promises, perhaps with corrections caused by noise or implementation errors. The promises might not drive the trajectories, but they constrain the observed behaviours depending on the degree to which they can be kept. This is what we mean by reliability.

Assumption 1 (Systems are non-deterministic but constrainable) *All systems should be considered non-deterministic, regardless of their design, since they interact with environments which cannot be predicted. Agents act non-deterministically, and they keep and receive promises non-deterministically. Thus, they cannot be predicted precisely, but their behaviours can be narrowed or focused into an acceptable range of values.*

Non-deterministic systems experience the arrival of events or 'random' (non-modelled) occurrences, some of which are acceptable to system policy and some of which are not. Events which are not acceptable may be called faults.

[2]This model can be applied whether an agent is a human or some kind of proxy, machine, device, etc. Ignoring the distinction makes sense because we humans experience any behaviour as if it were anthropomorphic or 'caused by agencies': think of 'the ghost in the machine', or 'Maxwell's daemon'. This is harmless, even practical, as long as we don't read too much into what we mean by an agent. Ultimately all agency stems from some human source, either by programming or direct control.

3.3.4 Design flaws resulting from missing promises

Promises are often made conditionally, in limited contexts. Sometimes a system designer does not make promises to cover all the possible contexts, and situations arise in which no promise was made. Agents in the system have no guidance on how to behave. The behaviour of the system is them *undefined*, and we cannot anticipate what will happen. Generally, as the scale of a system becomes large (large N, large size, etc) then we should expect more of these behaviours, simply because we have more and more incomplete information.

Comment 11 (Security exploits) *Many security exploits are based on the absence or mismatch of promises covering unexpected conditions.*

3.3.5 The trajectory of an agent making promises

When an agent makes promises, it is analogous to 'laws of motion' in physics. Laws of motion are not laws, of course, but observed norms. Promises that are kept most of the time lead to behaviours that are regular and characterizable as trajectories through some set of states.

A propagator that describes the change in a single agent's trajectory,

Definition 20 (Trajectory propagator) *Consider a promise binding between two agents S and R:*

$$\pi_+ : S \xrightarrow{+b_S} R \tag{3.4}$$

$$\pi_- : R \xrightarrow{-b_R} S \tag{3.5}$$

We define the propagation function to be the set valued overlap transfer of state - the state is the promise

$$\Delta_{\pi_\pm}(q, q') = \tag{3.6}$$

This is one kind of propagation of state in a system.

Another kind relates to interactions between agents. Constant promises lead to constant outcomes, so it is changes that invoke response, and lead to propagation.

3.3.6 Faults in communication

The most basic flaw in a cooperative system is the lack of a common language for mutual understanding (lingua franca). If agents, whether humans or machines, cannot make themselves understood to one another then it is not possible to make promises, or to assess their outcomes. Without promises, there cannot be expectation and cooperation will be ad hoc. Similarly, if there is only partial understanding, then errors or interpretation can lead to faults.

Comment 12 (Lingua franca) *The language of a promise must be shared between the promiser and the promisees. It need not be a spoken or written language: any kind of symbolism that conveys meaning will do, e.g. the shape and placement of a door handle, the form of a chair, the on/off symbol on an electrical device, standard road signs, etc., are all examples of languages for communicating intent.*

Example 8 *External circumstances may put agents into a mode where they lose their ability to understand the language of their surroundings. For humans, loss of comprehension gets gradually worse in high stress environments. For machines, there can be sudden and catastrophic breakdown of understanding as a result of changing the context of a single component, e.g. a version mismatch. Moreover, while humans can sometimes adapt to change, machinery (especially that which is designed with the assumption of a protected environment) is particularly exposed to error arising from intolerance of change.*

These scenarios result in a breakdown of the assumptions on which a system was built.

- An exchange relating to a promise, which cannot be understood, cannot be received as a promise by another agent.

- Agents may only understand part of a promise, if their dictionary of language or terminology is incomplete[Bur15].
- Words with multiple meanings (homonyms) can lead to misinterpretationsof what is promised.
- Shared or unstated assumptions (that which is taken for granted) might not be as universal as one expects.

Example 9 (IT system languages) *Language is a basic part of information technology. Common languages are assumed in protocols, software versions, user experiences and interfaces, encrypted messages, procedures, and icons.*

3.3.7 Shared assumptions

Trusted assumptions and implicit promises are a common cause of system design flaws. Assumptions about what is common knowledge can allow significant compression of communications.

Example 10 (Fuel types) *If an driver (part of a car system) believes that all cars run on petrol/gasolene, whereas a particular vehicle runs on diesel, an attempt to keep the promise to refuel the vehicle could result in a fault. If the car does not make a clear promise about what kind of fuel it can use (-fuel), then it might bind to any promise of type (+fuel). Today the fuel nozzle receptors for petrol and diesel are different, making much clearer promises about what type of fuel is expected.*

This is a trust issue. For example, the assumption that, once set, a system property is immutable and is relied upon not to change could save considerable monitoring communication. If the assumption is violated, them it becomes a straightforward failure mode: in this case, a design flaw, since assumptions are usually not promised.

Agents do not have to all share the same language, but they need to be able to communicate their without errors of comprehension.

Example 11 (Power supply) *Across large parts of the world, the electrical power is 240 volt A.C. In the North America and Japan, it is 110 Volts A.C.. The power outlets have very different shapes and connecters. The changes in power are sudden and often take travellers by surprise. This has led to many destroyed electrical devices.*

Example 12 (Character encoding) *A document written in a character encoding such as ASCII, EBCDIC, or Unicode is not compatible with a document written in a different encoding, and appears to be garbage.*

Example 13 (Protocol) *If one takes away all punctuation from a text, it becomes quite difficult to read, and may result in fatal misunderstandings. Take the well-known joke of the panda who enters a bamboo restaurant with an extra comma, and comes out angry with blood on his hands, having read: "Panda eats, shoot and leaves" instead of "Panda eats shoots and leaves".*

3.4 Agent interactions

An autonomous agent does not depend on any other; it has everything it needs, and is immunized against faults in other agents' promises. This is the value of autonomy. There is also value, however, in delegating to specialist services, which brings with it dependency and risk of fragility.

3.4.1 Cooperation faults arising from non-neutral promise bindings

If we can assume language is mutually compatible, then faults can still arise from incomplete intent. Promises have polarity:

- + promises express the intent to provide e.g. service delivery
- - promises express the intent to accept, e.g. access control rights

Both are pre-requisite for the expectation of service propagation from one agent to another.

Rule 1 (Promise graphs should be promise-neutral) *Promise bindings always consist of a (+) promise and a (-) promise:*

$$S \xrightarrow{+srv} R \qquad \textit{Service is promised} \tag{3.7}$$

$$R \xrightarrow{-srv} S \qquad \textit{Service is expected, and will be used} \tag{3.8}$$

The promise sum is neutral i.e. $\pm srv \simeq 0$.

A mismatch between positive and negative promises indicates a semantic flaw of design or implementation. If it does not currently exist (it might be accidentally true, but not intentionally true) then it will likely become a problem.

- **No + promise**: User expects a promise to be kept, but no promise was given.
- **No - promise**: A promise was given, but it was never used.

This provides a simple syntactic check, based on the model of promises.

3.4.2 Faults in interactions between agents

When a stakeholder or recipient R depends on the result of a promise sent by S, a system dependency has failed (see fig. 3.4). R must assess that its expectations of cooperation, by S, have not been met. The figure shows the

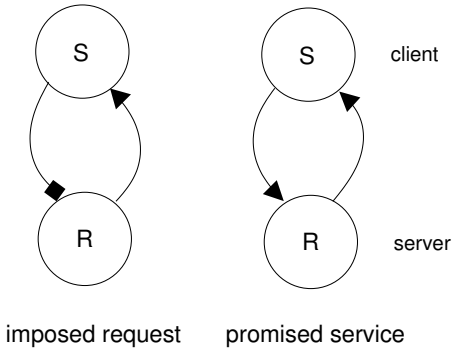

Figure 3.4: A dependency of one agent on the promise of another leads to broken cooperation.

contrasting cases for cooperation. The bindings may be:

$$S \xrightarrow{+b}\!\blacksquare\ R \tag{3.9}$$

$$R \xrightarrow{+b} R \tag{3.10}$$

i.e. S imposes its request for $+b$ onto the recipient R, and R answers with a promise to provide $+b$. This represents the usual tradition 'push button' thinking in technology. An imposition is understood to come from outside the agent. It contrasts with the voluntary promise relationship, where the promise always comes from within the agent:

$$S \xrightarrow{+b} R \tag{3.11}$$

$$R \xrightarrow{-b} R \tag{3.12}$$

i.e. S promises a service $+b$ to R (voluntarily, and without imposition), and R promises to accept it $-b$. What could be the reasons for non-cooperation for impositions? Here are some proposed failure modes:

1. S's imposition request was not sent.
2. S's imposition request was not received (S and/or R were available).
3. S's imposition request was not understood, or interpreted correctly.

4. R's promise to reply was not made, perhaps because the imposition was not received.
5. R's promise to reply was made but not kept.
6. R's promise to reply was made but not kept in time.

For promise relationships we have a symmetrical situation:

1. S's promise to deliver was not made, or was made but not kept.
2. S's promise was not received, or was not understood, or interpreted correctly.
3. R's promise to accept was not made, or was made but not kept.
4. R's promise was not received, or was not understood, or interpreted correctly.

Since promises are the sole responsibility of the agent making them, neither agent depends on the other for the making of its promise.

- The imposition version of this has the form of a client-server system (push).
- The promise version of this has the form of a publish-subscribe (pull).

Clearly, the number of failure modes is huge compared to what we are usually willing to invest in prevention or repair. This makes the design of systems that are fault tolerant seem attractive both as a practical and a potentially cheaper option.

3.4.3 Serial dependency versus parallel redundancy versus fault tolerance

A compromise is to design systems with redundancy and repair or key components. Seeking the balance between these different strategies is potentially a conflict of interest, representable as a type II model[Bur04a] or strategic game. We examined some basic cases using traditional structure function analysis in chapter 2.

There are three basic patterns:

- Modular dependency, or serial dependence of one module on another, such as software packages.
- Replica sets, or parallel redundancy to give failover in case of fault.
- Retry, or soft failure with tolerance.

We can define these cases explicitly.

Definition 21 (Serial dependency) *In a flow-based, transactional system, an error introduced by a dependency can affect the dependent agent, and all later agents. Once lost, information cannot be recovered. This is a high risk delegation.*

Definition 22 (Redundant dependency) *If a serial dependency has several alternative sources on which it can rely, it is less likely to be in a situation where one of them is unavailable, provided it promises due diligence in using the available services. This is a mitigation of risk.*

Definition 23 (Tolerance) *In a promise-based system, an error introduced into a serial chain can sometimes be absorbed, mitigated or eliminated by later actions, provided authoritative (template) information suitable for error correction can be kept locally.*

In the flow case, changes are relative, and we can prove that error correction is Byzantine or impossible. In the promise or policy system, changes are absolute, and we can define error correction by reset or zero-operation[BC11].

Example 14 (Fault tolerance - bullet proof vests) *renal failure sleep*

The sampling rate for agents working together plays a key role in causality and error propagation[BD07, Dis07]. All events that happen within a single sampling resolution are simultaneous, according to the next agent in a chain, hence rapid repair can lead to unnoticed errors. This is the approach used in memory CRC, processor errors, CFEngine, etc. Once faults escape confinement, they cannot be undone.

3.4.4 Redundant alternatives - mitigating a serial dependency

By reversing the signs in figure 3.17, we can average out unreliabilities, as one does in data analysis by repeating measurment samples. Agents can also measure the variability and warn about inconsistency. Unlike pure data values, semantic averaging may be considered unacceptable, when it has not been alllowed for.

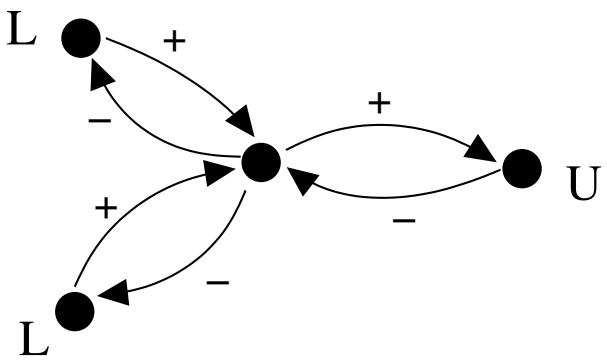

Figure 3.5: Convergence by natural aggregation and averaging

1. The multiple sources S_i all promise service $+d$ to the intermediary, within their accuracy tolerances.
2. Intermediary/relay I, promises to accept the values and select one, if suitable for meeting the dependence $s|d$.

$$S_1 \xrightarrow{+d} M \tag{3.13}$$
$$M \xrightarrow{-d} S_1 \tag{3.14}$$
$$\dots \tag{3.15}$$
$$S_n \xrightarrow{+d} M \tag{3.16}$$
$$M \xrightarrow{-d} S_n \tag{3.17}$$
$$M \xrightarrow{+s|d} R \tag{3.18}$$
$$\tag{3.19}$$

Now I has secured sufficient reduncancy to ensure that one of the promises will be kept. However, how do we know that the sources S_i are consistent? I needs a policy to select $+d$ from one or more of the agents, and possibly combine them. If errors are random, a majority or average should suffice. If errors are systematic, and repeatable,

What is not clear from this simple argument is whether the two alternative sources are exactly equivalent, or as good as one another. This is not readily expressible in a classical component analysis; in a promise model, we have the ability to look at what alternatives promise and measure expectations relative to those promises.

It remains up to the aggregating agent (user) to deal with inconsistencies. Even if we work very hard to build a system that has consistent redundancy, the promises cannot be guaranteed, so it becomes the responsibility of the selector or aggregator to keep promises consistently even when the dependencies vary unpredictably.

3.4.5 Semantic fault tolerance by averaging - requisite diversity versus redundancy

The previous section may also be characterized as a form of statistical averaging. Putting all your eggs in one basket is a fragile strategy. A mixed strategy (in the sense of game theory) is a fault tolerance strategy.

Go for non-discrete service levels....

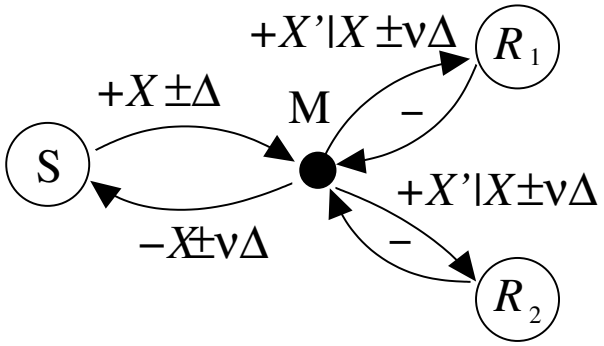

Figure 3.6: An error at the head of a chain of dependences can be absorbed by increasing accepted tolerances from $\Delta \to \nu\Delta$.

3.4.6 Serial fault tolerance: adding margins for error

Rather than propagating faults forwards, fault-tolerant agents can, in fact, absorb them, providing 'shock absorption' for errors, and correcting them to within the limits of their own tolerances (see fig. 3.6).

1. S promises service X, within its accuracy tolerance $X \pm \Delta$.
2. Intermediary/relay M, promises to accept the value in the range $X \pm \nu\Delta$, where $\nu > 1$.
3. Since $\nu\Delta > \Delta$, I accepts the service from S, and promises X' to R, hence there is recovery from a broad range of failures.

The assumption of a dependence means that there must be limits on ν too. If M could manage without the service from S, then why depend on it at all? Perhaps the tolerance of no service from S could be maintained by caching past results from S, with eventual expiry.

This shows that fault tolerant agents can keep promises, even when connected in series, by absorbing errors and faults upstream.

3.4.7 Tolerance of service inconsistency during selection from redundant parallel alternatives

The difference between agents might not simply be in performance or data (dynamics), but also in the details of behaviour (semantics). The equivalence of the two sources S_1, S_2 of a promised dependency assumes that the sources are sufficiently alike to keep the promise needed by the intermediary I. But what if the agents have different tolerances, accuracies or limits to their capabilities? We have to ask: how sensitive is the recipient to exact similarity?

Related to statistical averaging of data sources for resilience, software engineering sometimes wants to have global consistency. Dealing with inconsistencies in semantics requires a new policy: e.g.

- Picking a majority/quorum (Paxos etc) - sometimes called multi-master
- Picking the most recent value (vector clocks).
- Checksum or certificate verification and rejection

If we want system *fault tolerance* rather than potential for failing to reach majority agreement, it's clear from the foregoing example that we should tolerate inconsistency rather than try hard to ensure it, as downstream absorption is still the most robust strategy for keeping the total promise.

Comment 13 (Consistency) *There are fundamental reasons why strong consistency is not achievable in the general case. However, the idea of a quorum between servers, i.e. a minimum number of sources that must agree on an outcome in order to make a valid inference, is a popular methodology, especially in databases.*

All kinds of arbitrary and artificial strategies to claim truth are used for quora: e.g. using an odd number of servers (at least three) so that there can be a majority. None of these guarantees that the majority result is actually 'correct', unless we can verify what correct is promised to mean.

3.4.8 Convergent local repair

Repair is an agile method of fault tolerance. If a fault develops in theory, but has not been tested in practice, then there is a window of opportunity in which it may still be kept. Thus a sufficiently rapid repair can be an effective way of keeping promises. Some systems, e.g. agile fighter aircraft, are designed to ride the edge of stability in this way, as it affords them far greater maneuverability on a shorter time-scale than a more stable design would.

By embedding compressed information about intended state into a system throughout, repairs can be made in real time[Bur03, Bur04a, SW49]. If agents along a chain experience errors that can be self-corrected (e.g. Shannon error correcting theorem, Hamming codes, etc), a feedback loop may be applied to keep the total promise of the system, within a time T_{repair} (MTTR). This requires the presence of a model (and thus potentilly an agency responsible for providing and updating the model also).

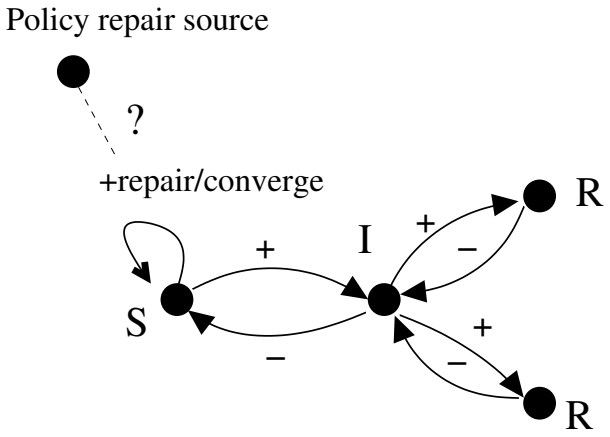

Figure 3.7: Convergence by feedback mitigates faults at the head of the chain, before amplification

Inline repair, or agents in the chain, can be appropriate or even cheaper or faster than redundancy if T_{repair} (MTTR) is less than the time it takes to acquire, aggregate and select a value from redundant sources (see section 3.4.4).

T general strategy introduced and used by CFEngine is to keep all agents as close to a policy state as possible at all times, then hope for fault tolerance.

Comment 14 (Continual repair is continual delivery of change) *Information re-injected into the system could be:*

- *Fixed information: repair to original state.*
- *New information: course correction, repair to improved state.*

The absorption of errors is the essence of system testing *before release. This kind of tight loop is the approach used in continuous delivery[HF10].*

3.4.9 Partially ordered promises

A chain or sequence of ordered promises is a fragile structure. Ordered promises form chains of dependencies. If a single promise fails, all subsequent dependees must also fail.

3.5 'Push versus pull' transactions in causal influence

'Push versus pull' is one of the choices systems designers have in designing cooperation[3]. In a push method, the source is favoured as the deciding agency (like a brain model). In pull methods, the receivers are favoured as the deciding agencies (like a cooperative society model). The law of agent autonomy would thus seem to favour pull methods over push: push must be imposed, with the intent to violate a receiver's autonomy. We shall explore whether this is indeed the case in detail. This is not a moral issue, but one that affects our ability to make a consistent interpretation of the transmission.

3.5.1 Definitions of push and pull

The definitions of push and pull do not refer to complete methods for communication or influence; rather, they characterize aspects of complete descriptions.

Definition 24 (Push) *A method in which a sender attempts to impose an influence onto a remote agent. The sender is the source of the intent to tranfer influence or information. This is an attempt to induce cooperation, or ignore autonomy:*

$$S \xrightarrow{-b}\!\!\blacksquare\ R \tag{3.20}$$

For this to succeed, without overwhelming force, there must also be a promise expressing a willingness to accept the imposition:

$$R \xrightarrow{-b} S. \tag{3.21}$$

Definition 25 (Pull) *A method in which a receiver voluntarily subscribes to the influence of a remote agent. The recipient is the source of the intent to transfer influence or information. The agents always retain full autonomy:*

$$R \xrightarrow{-b} S. \tag{3.22}$$

For this to succeed, there must also be a promise offering a service to accept:

$$S \xrightarrow{+b} R. \tag{3.23}$$

Both push and pull methods for cooperation can be analyzed using promises and impositions (see fig. 3.8). Neither construction is a complete description of a system, so the properties of these approaches depend on what other promises are made around them.

3.5.2 Properties of push and pull

If we draw a super-agent boundary around agents that are transmitting influence either by push or pull, what can be say about the components' reliabilities, i.e. the fidelity of that particular constellation of agents?

- *Speed of response (agilty)*

  The speed of response depends only on the path length and processing time of the agents, which is independent of push and pull.

  In a push model, a message from source can be imposed immediately without delay, but the recipient might not be awake, or have time to pay attention to it straightaway.

$$R \xrightarrow{-b} S \tag{3.24}$$

$$S \xrightarrow{-b}\!\!\blacksquare\ R \tag{3.25}$$

$$R \xrightarrow{+rsvp|b} S \tag{3.26}$$

$$*S \xrightarrow{-rsvp} R \tag{3.27}$$

[3] It is a general prejudice arising from our manual experience to think of push as a driving force for change, but we know from countless examples that both possibilities exist. Gravity, magnetism, 'vacuum pressure', etc are all examples of pull. Even at the elementary particle level, interactions can be framed as retarded or advanced propagation, depending on how we choose to fix boundary conditions. It is precisely this desire to fix the end state, rather than the start, which favours pull.

The star shows the point at which the sender may claim awareness of the outcome. This has one prerequisite. Because the outcome happens at a location R that is inaccessible to the source of the intent S, for confirmation of success, the recipient R must promise a confirmation of receipt service 'rsvp'. Only when this notification of a response is accepted does the sender know the outcome of the imposed push.

In a pull model, the rate at which an recipient polls the source for updates might be fast or slow, but it happens at its own behest, so it is ready to the update by definition.

$$S \xrightarrow{+b} R \tag{3.28}$$

$$*R \xrightarrow{-b} S \tag{3.29}$$

The star shows the point at which the sender may claim awareness of the outcome. This time, there are no prerequisites for this to happen, and the outcome is immediate on initiating the pull. Not the provider does not know about the state of the recipient (it may not need to, since it has no stake in the intent to acquire the service), so one could make the same kind of monitoring promise from recipient to server, but this is no longer a necessity.

Given the uncertainties in both cases, it is impossible to say whether push or pull might be faster than the other. This depends on the circumstances. What we can say is that the push case is not guaranteed to affect the recipient (if it is not awake or willing to accept the push). In the pull case, the acceptance of a message is implicit in the attempt to subscribe to it, but the source might not be available when the recipient wants to pull.

Lemma 2 (The speed of response for pull vs push) *The speed of reponding to an intended outcome is strictly indeterminate for both push and pull. Either could be faster under particular circumstances.*

PROOF

- *Is the initiator (source of intent) able to know the outcome (success/failure) of the cooperation?*

  In the case of the push, the initiator is the source, and the result is the recipient, thus intent and result are at opposite ends of the operation to obtain the result. In the absence of further promises to feedback the result, the initiator cannot know the outcome of the operation. This is sometimes called a 'shot in the dark' or 'throwing it over the wall'.

  The initiator may promise to monitor the result, if the recipient promises to allow it as a service. Thus we may add two additional promises (with corresponding uncertainties) to determine whether the push succeeded. The recipient can, of course, lie. The source's distance from the point of compliance is a fundamental problem for verification. Indeed, if the receiver intentionally did not accept the push in the first place, it is unlikely to report back truthfully either. Thus, a push scenario is inherently unknowable.

  In the case of pull, the initiator is the recipient, and it is the affected party. Thus there is immediate confirmation of success or failure. This represents maximum possible certainty. It is impossible to lie about the success of the promise.

Lemma 3 (The uncertainty of pull vs push) *The uncertainty of an intended outcome is strictly less for pull than for push.*

PROOF

- *Can conflicts be resolved if multiple sources have different intended outcomes for a single recipient target?*

  If several agents try to push to a single recipient, the result is inconclusive, and potentially order dependent. This is the multithreading contention problem. Serialization may be provided by locking, or the requests could be separated into independent non-shared contexts, if the recipient promises to support that. The context for each push is computed by the source agents, and is unknown to the recipient, and hence cannot be reproduced later for adaptation or recall by the recipient.

If an agent tries to pull from multiple sources, the result can be intentionally serialized by the recipient, and either integrated into a single context, or separated into reproducible, independent contexts. The contexts are computed by the instigating, pulling recipient and thus may be reproduced later for adaptation.

This problem is considered further in section 3.5.5.

Lemma 4 (The locality of pull vs push) *All information in a pull scenario is local to the agent affected by the intent, hence the initiator knows intent and outcome immediately. Information in the push scenario is delocalized: the initiator is not the affected agent, and the outcome is not automatically known to the initiator.*

PROOF

- *Can failures be repaired?*

  A pull agent has full information about what resources it is missing to keep a promise, so it can attempt to acquire this from any agent that can keep such a promise. A push agent does not know what is missing from the recipient agent, so it can only shoot resources blindly[4].

- *Security?*

  TODO

3.5.3 Situation awareness in pull and push

The question we seek to ask here is: to what extent are we are able to claim reliability of a push or pull method, within a system? We see that pull-based approaches have several advantage over push based methods. In section 3.5.5 we will also see how pull resolves consistency conflicts, or so-called split brain problems.

There is a fundamental motivational conundrum for push-based (imposition) controls: how does one motivate an external intervention in a system without priot reliable information? A random imposition could be a shot in the dark. Remote control thus has a bootstrap problem for the awareness of state.

In a local scenario, where information about system state is both available and trusted, the motivation for an intervention or repair is always present; but, in a remote control scenario, information is not close at hand. This means:

- We have to trust the source of information about remote state.
- Information has to travel quickly enough to be current.
- Information has to be separable from other remote effects in order for the remote agent to understand the possible consequences of an imposition (shot in the dark).

Thus the problem of motivating remote interventions is risky, the father away from the point of action one initiates intent.

Lemma 5 (Locality and the completeness of information) *All information in a pull scenario is local to the recipient agent. Information in the push scenario is delocalized.*

PROOF...

3.5.4 The relative stability of push and pull

Consider the representation of a single atomic agent making promises to push and pull influence (e.g. data, material, etc) in figure 3.8. The characteristics of these arrangements seem to have properties that are a natural fit for their topologies. The topology of a push, from sender S, suggests dissemination, amplification from one to many and delocalization of influence. The topology of pull, to receiver R, suggests localization and the ability to resolve multiple influences by personal selection from many alternatives to one result. The opposite scenarios also warrant attention (see section 3.5.5). Pushing from many to one would result in conflict at a point, but pulling from one to many is a common form of dissemination sometimes called publish-subscribe[5].

[4] This approach is sometimes used in systems: continuous mandatory replacement of parts to ensure correctness. It can be effective, but it is disruptive and wasteful.

[5] This is a method of dissemination used in magazine stands, bookshops, content delivery networks (CDN), supply chains, etc.

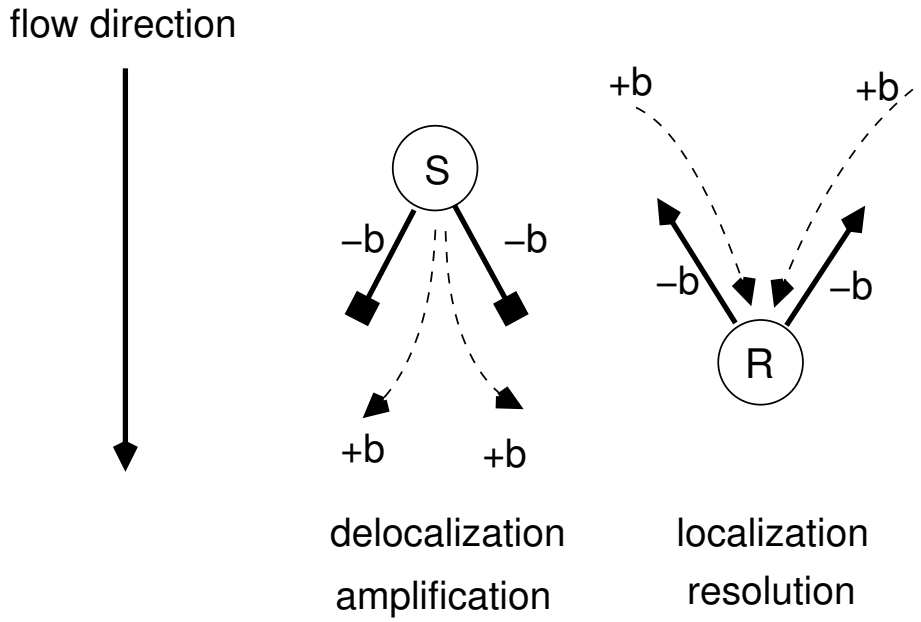

Figure 3.8: Push and pull methods in relation to a single agent. The geometry of a push, from sender S, suggests dissemination, amplification and delocalization of influence. The geometry of pull, to receiver R, suggests localization and resolution by selection from alternatives. Note the direction of intent is marked by the promise arrows, while the direction of service flow is indicated by the downward arrow.

Comparing the left and right hand characteristics, a number of things is immediately apparent. The left hand side shows a push method, in which the service source S pushes out by imposing b onto some number of recipients. The dotted promise lines indicate the alternative promise approach. The right hand side shows upward promises from the recipient to pull $-b$ from one or more sources. In both cases the diagrams are shown from a single autonomous agent, in other words, what we are comparing in the diagram with what a single agent can accomplish without cooperation.

Some more characteristics are summarized in table 3.2.

POLARITY	NOTES / CHARACTERISTICS / ASSOCIATIONS
PUSH	BRAIN MODEL, FLOODING, BROADCASTING SINGLE SOURCE (VERTICAL SCALING) SPONTAENOUS IMPOSITION (SURPRISE), FLOW-DRIVING, AMPLIFYING ASSUME SUCCESS, NO FEEDBACK, DIVERGENT NON-LOCAL OUTCOME IS UNKNOWN TO INITIATOR (DECALIBRATION)
PULL	SOCIETY OR CONSUMER MODEL, SUBSCRIBING (TO PUBLISHED MATERIAL) MULTIPLE REDUNDANT SOURCES (HORIZONTAL SCALING) PLANNED PROMISES (PREDICTED), COUNTER-FLOW, AGGREGATING, SELF-CONFIRMING OUTCOME, IMMEDIATE FEEDBACK, CONVERGENT LOCAL OUTCOME IS KNOWN TO INITIATOR (CALIBRATION)

TABLE 3.2: COMPARING GENERIC ATTRIBUTES OF PUSH AND PULL.

Comment 15 (Confirmation of outcome) *When agents provide a confirmation of outcome as a service, e.g. return codes from software functions, they do so as an additional promise, that may or may not be kept. Confirmation does not necessarily improve our knowledge, because we have to trust the information. If an agent is unreliable in keeping a promise, why would we expect it to be any more reliable in reporting the outcome?*

3.5.5 Resolving conflicting dependencies with push and pull (split brain problem)

Let's return to the issue of confict resolution, or how to assimilate redundant promises consistently. In imposition systems and push systems, a major source of problems is inconsistency, as there is contention over shared reosurces (see figure 3.9). One way to avoid the problem of contention with impositions is to ensure that impositions only

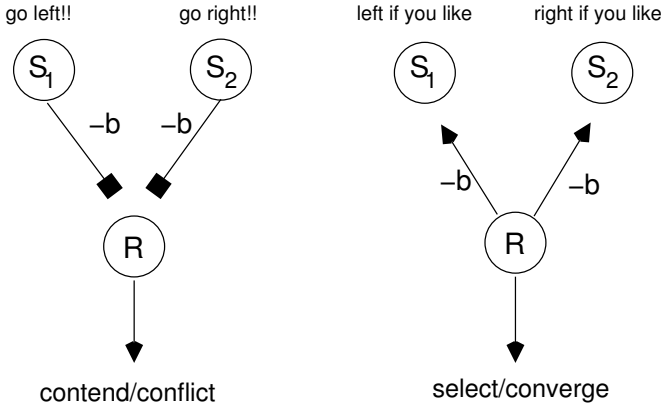

Figure 3.9: In the imposition case, one of the agents has to be at odds with its intent. In other words, expectations cannot be met, and the outcome must fail unintentionally. In the promise case, all expectations can be met and all agents have complete information about the expected outcome. Any deviation is now due to non-intended issues.

cause divergent outcomes. If each imposition lives entirely independently, within a private context, there is no problem (see figure 3.10).

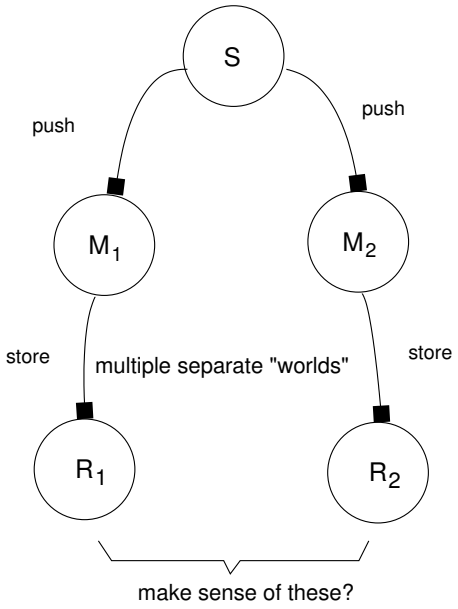

Figure 3.10: If influence is pushed redundantly into completely separate spaces, there can be no conflict or doubt within each separate instance of these 'many worlds'.

Example 15 (Private world consistency) *In information technology, client-server systems, where resources are pushed to a single queue, or which potentially in conflict with shared resources, often 'fork' their processes into privately segmented workspaces. These erect separate agencies where there is no contention. If the segmented threads still need to work with a shared resource, locks requests are used to force the imposition processes into a requesting 'pull' mode of operation.*

In the example above, a principle is used to escape from contention, which is well known as a resource lock in IT systems. Think also of the lock on toilet stalls which serializes access to the shared resource by multiple users. We make sense of contention in shared resources by having a single agent select a result from a collection of alternatives. This is the transition from the lefthand side to the righthand side in figure 3.8. Agents can avoid many

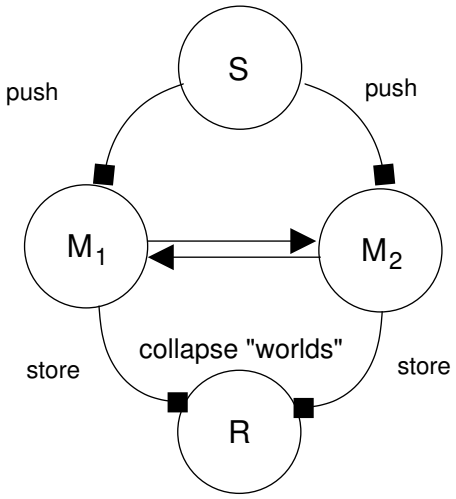

Figure 3.11: Conflict and inconsistency can result when we collapse independent worlds into a single picture. We do this when we average data in statistics. The arithmetic mean is an algorithm for resolving the inconsistencies of independent measurements. Voting or quora are another approach for odd-numbered collections of outcomes. In each case, resolution is obtained when the observer chooses (pulls or accepts with a (-) promise) a representative value from the values ofference.

of the problems associated with push or impositions by reversing their thinking to a service oriented subscription access approach.

Comment 16 (From push-commit to publish-subscribe) *For essentialy cultural reasons, the push method is ubiquitous in all forms of technology, as well as in society. We like to treat the world as if it were an extension of our bodies. However, publish-subscribe methods (otherwise called 'pull') are becoming increasingly popular, as they resolve many issues (especially for web scale), such as allowing tolerance of inconsistency, or so-called 'eventual consistency'. It is often argued that we must have consistency, but often this is overstated, and the regions over which consistency is needed are much smaller than systems generally try to apply them over. Pull methods allow consistency on a need-to-know basis.*

3.6 Propagation of influence

When promises are conditionally chained together in space or time, they can lead to cooperative processes that span multiple agents. Processes may therefore also be described and documented using the language of promises. Transmission of intent leads to the propagation of influence, both good and bad. This can lead to sudden cascade failure modes, whereby systems breach catastrophically and beyond repair—an indication that the stability influence propagation needs careful attention, and prevention of failure by fault tolerance is an important strategy.

3.6.1 Rate of fault propagation

To understand the rate of propagation of faults, we need to understand the promise bindings that lead to propagation. A promise binding has two parts:

$$S \xrightarrow{+b} R \tag{3.30}$$

$$R \xrightarrow{-b} S \tag{3.31}$$

There are several relevant timescales to consider:

- The rate at which a + promise is kept, i.e. changes in promise status take place. (MTTR)
- The rate at which changes happen to affect promise keeping (MTBF)
- The rate at which a - promise samples the results of the + promise.

In a steady state behaviour promise bindings are equilibria, but during transient activity, the intial behaviour is sensitive to these timescales. The result is called a race condition, and the outcome can depend quite sensitively and unpredictably on the particular circumstances.

- A + promise might be kept too late to be useful to the agent that relies upon it with its - promise samples the outcome before it has had time to keep its promise.
- A broken promise or fault might lie in wait for an extended time before it is relied upon and triggers a consequence that causes an observable effect. Thus the propagation rate might be significantly slower than the MTBF at the 'root cause site'. Similarly, once a critical collapse has been triggered, the catastrophe failure mode will travel at its own rate.

Example 16 *In the Estonia ferry disaster of 1994, the door to the car deck was not closed properly. Water did not start to enter the ferry destabillizing it until hours later when the sea conditions passed some critical threshold. Had the promise to keep the door closed been kept the failure threshold would have been immunized.*

3.6.2 Separation of dynamical and semantic outcomes

There are two kinds of networks

- Adjacency of agents (spacetime), resource transport networks (Markov)
- Cooperative processes (with valued outcomes) (Prerequisite)

A semantic outcome may happen on a different timescale to the underlying processes that lead to it.

Misinterpretation of the semantics of a failure. If we don't have an extensive causal understanding of the pathways of influence, we could miss possible predictions.

3.6.3 Promise trajectories

When a promise $\pi = \langle S, R, b(\tau, \chi_\tau)\rangle$ is kept, the constraint χ_τ, belonging to the promise body, expresses a restriction on the state q_τ. To elaborate on this matter, more rigorously, we need a more detailed model of agent state, by imagining sets of microstates q_τ that refer to the variables inside agents that pertain to promises of type τ. There are various possible representations of this using functions, and using matrices that we need not go into here (see [BF07, BF08], for instance). The key features of such a representation can be described as follows.

Given a promise $\pi = \langle S, R, b(\tau, \chi_\tau)\rangle$, and a set of states that may be associated with the promise q_τ, we associate, with the enforcement of the promise, a set of operations $\hat{O}_\tau(\chi_\tau)$ such that the action of the operator on a general state propagates the agent from the old to new state $q_\tau \to q_0$ that complies with the restriction χ_τ, whose orbit is the set of promised states:

$$\chi_\tau \;:\; q_\tau \in \{q_\chi\} \tag{3.32}$$

$$\hat{O}_\tau(\chi_\tau) \;:\; \hat{O}_\tau(\chi_\tau)\{q_\chi\} = \{q_\chi\},\ \forall q_\chi. \tag{3.33}$$

$$\hat{O}_\tau(\chi_\tau) \;:\; \hat{O}_\tau(\chi_\tau) q_\tau = q_0 \in q_\chi. \tag{3.34}$$

Thus, another way to represent this, is to say that the operation $\hat{O}(\chi_\tau)$ effectively generates a function f_π that is absorbing for all its arguments to a fixed domain: $f_\pi(q) \to \{q_\chi\}$. This is called a convergent operation.

The introduction of a promise, or a change to an existing promise, can alter the trajectory of the state $q_\chi(t)$, i.e. the sequence of states the agent moves through, which, in general, is an orbit or function of time. The relationship between promises and agent trajectories was described in [BF07, BF08]. For differential changes the resulting properties of motion closely resemble Newton's laws for particles, as they must, where an effective force $\hat{F}$ can be related to a complete promise binding $\delta\hat{O}$. The time-development of the trajectory leads to one kind of propagation of influence. The change of the promise that guides the trajectory leads to another, known as the response function (see section 3.6.7). For the purpose of these notes, all we need to know is that:

- Promises are kept by associating convergent operations, that are convergent and idempotent, when acting on local agent states[6].
- The repeated verification and implementation of a promise's constraint may be carried out by repeated application of the operator or function on the subset of states, within the agent, that refer to the promise.
- The resulting state, or sequence in time, is the promised τ-trajectory of the agent. This represents its observable behaviour.
- A change in the promise that selects this trajectory can only be made by the agent itself, or by the agent promising to accept changes from an external source (another agent). Such a change may be regarded as the propagation of influence over the agent's trajectory by the source, in the form of a conditional promise that depends on the source.

3.6.4 Propagation of influence

If we assume that two agents are able to communicate for the purpose, the flow or propagation of intent from a point of origin to other agencies in a system involves cooperation between its component agencies.

Let's look at examples of transmission from Sources S through interMediary agents M to Receivers R, with graphical nomenclature:

$$S \to M \to R \tag{3.35}$$

We assume that the services promised by these nodes are refreshed on a continuous basis, or with some fixed time scale, so that we can speak of time to repair a promise not kept. We cannot take it for granted that influence propagates. For example, suppose we consider the promises:

$$S \xrightarrow{+\text{I am sad}} M \xrightarrow{+\text{I am happy}} R \tag{3.36}$$

There are two promises between three agents. For propagation to occur we would have to be able to say that the promise from S influenced the promise from M. But, as written, there is nothing to indicate that the cause of M's asserted happiness has anything to do with S's asserted sadness. In fact, M has not even promised to care whether S is sad or not.

To encode that, we would have to introduce conditionals, or a functional dependence, and acceptance promises:

$$S \underset{+\text{I am sad}}{\overset{-\text{You are sad}}{\longleftarrow\!\!\!\longrightarrow}} M \underset{+\text{I am happy}|\text{You are sad}}{\overset{-\text{You are happy}}{\longleftarrow\!\!\!\longrightarrow}} R \tag{3.37}$$

The shorthand for this is the notation

$$S \xrightarrow{+\text{I am sad}} M \overset{-\text{You are happy}}{\longleftarrow\!\!\!\xrightarrow{+\text{I am happy(you are sad)}}} R. \tag{3.38}$$

Now we can say that the reason why M is happy is in response to S's sadness. The state of S has influenced the state of M. R is influenced by M's happiness in promising to accept it, but the influence goes no further.

Example 17 *A less frivolous example helps to underline the point:*

$$\mathit{Outlet} \xrightarrow{+\mathit{power}} \mathit{Light} \overset{-\mathit{light}}{\longleftarrow\!\!\!\xrightarrow{+\mathit{light}(\mathit{power})}} \mathit{Person}. \tag{3.39}$$

Here we have an agent promising power (a power outlet), and a light bulb accepting this (by plugging into it), and the result is a promise of light, which is accepted (perhaps by switching it on). Unlike models of electricity, a promise model does not assume an inevtitable flow of current, or an inevitable flow of influence.

[6] A proper description of these states is beyond the scope of this work. This is not a completely trivial matter. Any dynamical system, in general, is characterized by two 'canonical' quantities: q and δq at each epoch.

The model of pre-conditions encodes what we mean by causation, or propagation of influence. Causation is a special case of propagation, because it is purely dynamical. It requires a chain of dependencies:

$$S \xrightarrow{+a} M_1 \xrightarrow{+b|a} M_2 \xrightarrow{+c|b} \ldots . \tag{3.40}$$

To complete this, we have to use the conditional promise law to quench the requirements and complete the cooperation[BB14].

Comment 17 (Conditional promise quenching) *Recapping, from basic promise theory: a local conditional promise, is equivalent to a non-conditional promise, if the condition $+b$ is also promised to be 'true', i.e. satisfied:*

$$A_1 \xrightarrow{T(+b),S|b} A_2 \equiv A_1 \xrightarrow{S} A_2 \tag{3.41}$$

If the condition is only promised independently (possibly true or satisfied), then

$$A_1 \xrightarrow{+b,S|b} A_2 \simeq A_1 \xrightarrow{S} A_2. \tag{3.42}$$

A non-local conditional promise, is equivalent to a non-conditional promise, if a promise to acquire the dependency $-b$ is given:

$$A_1 \xrightarrow{-b,S|b} A_2 \equiv A_1 \xrightarrow{S(b)} A_2 \simeq A_1 \xrightarrow{S} A_2 \tag{3.43}$$

Since the latter is the usual case, we give it the special quasi-functional notation in the promise body to reduce two promises to one.

Using the result in the box above, it is straightforward to show that, in order for a chain of conditional promises to propagate some property τ, the chain must stack up dependencies, accumulating points of failure along the way. Relaying through 'middlemen' makes a system more fragile. Starting from a non-conditional edge or boundary node A_0:

$$A_0 \xrightarrow{b_0} A_1 \xrightarrow{b_1(b_0)} A_2 \xrightarrow{b_2(b_1(b_0))} A_3 \xrightarrow{b_3(b_2(b_1(b_0)))}, \ldots \tag{3.44}$$

And, the bindings are completed by

$$A_0 \xleftarrow{-b_0} A_1 \xleftarrow{-b_1} A_2 \xleftarrow{-b_2} A_3 \xleftarrow{-b_3}, \ldots \tag{3.45}$$

For causal linkage, only the final link in the chain has to be keep a promise of τ; but, if we want to speak of propagation of τ, e.g. propagation of consistent information, the it has to be the same information. The pre-conditional chain is sufficient to claim a causal pathway no matter what the previous promise types. Note that, because there is no identical equivalence between non-determinstic outcomes, we cannot say that a promise to propagate influence necessarily reduces to a produce of the propagation over the links. At each scale, there are irreducible propagators that are independently assessable, e.g. the propagators with retarded (causal) boundary conditions:

$$\Delta_>(S,R) = \Delta(R|S) = S \xleftrightarrow{\pm b_0} R \tag{3.46}$$

$$\Delta_>(S,M_1,R) = \Delta(R|M_1S) = S \xleftrightarrow{\pm b_0} M_1 \xleftrightarrow{\pm b_1(b_0)} R \tag{3.47}$$

$$\Delta_>(S,M_1,M_2,R) = \Delta(R|M_2M_1S) = S \xleftrightarrow{\pm b_1} M_1 \xleftrightarrow{\pm b_1(b_0)} M_2 \xleftrightarrow{\pm b_2(b_1(b_0))} R \tag{3.48}$$

3.6.5 Promise types that propagate intent transitively

There are more examples of propagation, relating to semantics, or transmission of meaning, by what we call the transitive property. The rules of semantic transitivty were discovered by A. Couch[CB09], in relation to semantic inference.

- If C is affected by B and B is affected by A, then C *might be* affected by A.

The latter expresses both potential 'tolerance' and potential uncertainty.

- If C carries B and B carries A, then C carries A.
- If C encapsulates B and B encapsulates A, then C encapsulates A.

Note that, in this case, there is no sense in which and agent carrying another agent depends on the being carried by another, so we cannot require it as a precondition. This is a different kind of propagation. Note also the subtlety of meaning. It might not be true that if C contains B and B contains A then C contains. That depends on what you mean by contains. From whose perspective?

This underlines how making influence travel from one agent to the next is straightforward by mutual agreement; but, making it travel through multiple agents, without changing character altogether is far less certain. So there are two independent issues that we can separate:

1. How to define propagation of independent channels of influence between directly differential (adjacent) agents.
2. How to sum up the semantics of influence over longer paths (the integral problem).

In both cases, promises help to define the problem.

Note that, when we talk about promise agents, it is *assumed* that they try (continuously, by repetition) to maintain the state of promise compliance, not merely in a single event. In other words promises are timeless, unless bounded in duration relative to other promises. They do not assume that a promise that was kept in the past is still kept in the present.

3.6.6 Conditions for extended propagation (chains and processes)

The transmission of intent (and loss of intended behaviour or faults), through a system, is a network problem. The presence of a fault in a localized agent can be transmitted onwards through the system, to dependencies, with widespread semantic and dynamic repercussions.

1. On each link, the $\pm$ promise tyoes need to match
2. On each link, the bodies $+b$ and $-b$ need to have non-zero overlap.

The propagated effect is proportional to the overlap, if and only if the forward and reverse types are the same. These properties can be be encapsulated into a generalized response function for neighbouring agents, that takes into account parallel scaling.

3.6.7 The instantaneous response function

Consider now a function that is at the heart of propagation of influence between agents. It explains how promised behaviours can spread, how agents influence one another, and change one anothers' promises and trajectories. This is the generalization of the well-known linear response function from hydrodynamics or electrodynamics, in which we add the labelled semantics of the promises. It measures broadly the impact of a promise made at source, in a set of agents that rely on it 'downstream'.

Let $\pi_{ij}^{(\pm)}$ be the $\pm$ promise matrices, and $\Pi_{ij}^{(\pm)}$ be the promise adjacency matrices[Bur15] for the full promise interaction (see figures 3.15, 3.12, 3.13). It helps to define a few derivate parts of a promise, since a promise is a tuple, and we often only need parts of it. The set valued components of a promise bundle[BB14]:

$$\pi = \{S\} \xrightarrow{b} \{R\} \tag{3.49}$$

$$b = \langle \tau, \chi \rangle \tag{3.50}$$

may be written:

$$\begin{aligned} b(\pi) &= b && (3.51)\\ S(\pi) &= \{S\} && (3.52)\\ R(\pi) &= \{R\} && (3.53)\\ \tau(\pi) &= \tau(b(\pi)) = \tau && (3.54)\\ \mathrm{sgn}(\pi) &= \mathrm{sgn}(b) = \mathrm{sgn}(\tau) \in \{+,-\} && (3.55)\\ \chi(\pi) &= \chi(b) = \chi. && (3.56) \end{aligned}$$

With these definitions, we may now measure the number of agents affected by neutral promise bindings of any particular type of promise τ, according to what proportion of what is promised to them they choose to accept.

Definition 26 (Response function) *Consider two sets of agents: S_i (the sender set) where $i = 1, \ldots, s$, and R_j (the receiver set) where $j = 1, \ldots, r$, and let there be non-square promise matrices π^+_{ij} for promises from S to R, and π^-_{ji} for promises from R to S. Then we may defined the response function for transfer of intent from $\{S\}$ to $\{R\}$ by*

$$R\left(S, R, \pi^+, \pi^-\right) = \frac{1}{S} \sum_{i=1}^{S+R} \sum_{j=1}^{R+S} \Pi^+_{ij}(S,R)\Pi^-_{ji}(R,S) \left| \frac{\tau(\pi^+_{ij}) \cap -\tau(\pi^-_{ji})}{\mathrm{sgn}(\tau(\pi^+))\tau(\pi^+)} \right| \left(\chi(\pi^+_{ij}) \cap \tau(\chi^-_{ji})\right) \qquad (3.57)$$

where $|x|$ is the cardinality of the set x. The dimensions of $R()$ are the dimensions of χ.

The notation of the definition looks intimidating, but it is straightforward. The promise matrices $\Pi^{\pm}$ describe the graph of $\pm$ promises, in a neutral binding, from the sender set to the receiver set (see figure 3.15). These need not be balanced. The product of the forward matrix, with the matrix of promises in the backwards direction, selects only those combinations of agents that have promises in both directions. Roughly speaking, this has the form:

$$\text{Response} \propto \text{channel width} \times \text{type-binding} \times \text{agreed transfer} \qquad (3.58)$$

The term involving the promise type τ is always positive and dimensionless, and non-zero only if the reverse promise type is $-\tau$ when the forward promise type is τ, required to form an acceptance binding. Finally, the term involving the body constraint χ has the magnitude of the overlap of constraints between what was sent (+) and what was received (-). In other words, this is magnitude of the accepted transmission, analogous to the mutual information[CT91].

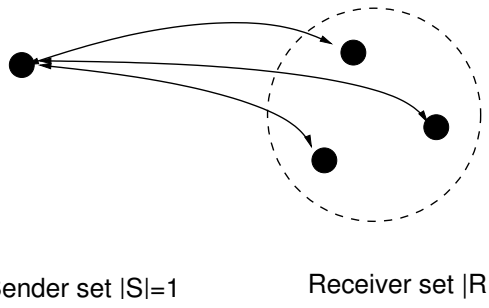

Figure 3.12: (Amplifying) gain in transferring promise bindings from one superagent set $S = 1$ to another receiver set $R = 3$ is proportional to the number of complete bindings and their overlap strength. In this case, if we assume the bindings are equal, the gain would be 3 times the strength of each binding.

Comment 18 (Complicated response function?) *The apparent intricacy of the definition of response exposes the fragility of response in a system where both dynamics and semantics play a role. There are three major failure modes:*

- *Missing or incomplete binding $\pm$.*
- *Mismatched promise type (input validation).*
- *Insufficient overlap between provider and user (input validation).*
- *Insufficient redundancy or channel capacity.*
- *Accidental positive gain, or misconfiguration leading to unintended amplification.*

The response function as designed applies for a static snapshot of a system configuration, and does not allow for the fast that the agents and promises might be in a state of flux. Thus we call it the instantaneous response. If we define a system real time t, then it could be added to the promise quantities $R\left(S(t), R(t), \pi^+(t), \pi^-(t)\right)$ to account for time-varying dynamics. Although a single snapshot has a linear form, the effect might be non-linear due to the non-deterministic nature of promise keeping, and effectively changing topology.

3.6.8 Propagation of uncertainty

Uncertainty in these bindings can occur from both ends. It is also useful to refer to the error function, for characterizing the degree of variablility in the behaviour of the agents. The error function combines the resulting 'error' or variation from a number of sources $a, b, c, \ldots$:

$$\mathrm{err}(a, b, c \ldots) \tag{3.59}$$

In the case of independent random errors, this reduces to the normal Gaussian Pythagorean expression:

$$\mathrm{err}_G(a, b, c \ldots) = \sqrt{a^2 + b^2 + c^2 + \ldots} \tag{3.60}$$

By using continuous variability in the keeping of promises, we may choose the threshold at which the failure to meet tolerance expectations is exceeded, and a variation becomes a fault.

If an agent is fault tolerant, then it can, in principle, absorb faults and propagate a non-faulty response forward. If not, then a fault will propagate by omission of an accepted promise kept.

Lemma 6 (Quantitative reponse) *The propagated effect of a promise π from S to R may be written $\Delta_\pi(R|S)$, written in a pre-conditional notation:*

$$\pi_+ : S \quad \xrightarrow{+b_S \pm \delta_S} \quad R \tag{3.61}$$

$$\pi_- : R \quad \xrightarrow{-b_R \pm \delta_R} \quad S \tag{3.62}$$

$$\Delta_{\pi_\pm}(R|S) = \quad S \xleftarrow{(b_S \cap b_R) \pm \left(\sqrt{\delta_S^2 + \delta_R^2} \cap b_R\right)} \!\!\!\!\!\!\rightarrow \quad R \tag{3.63}$$

The recipient is the ultimate arbiter of what is propagated, hence the final expression is limited by b_R

Thus, instead of the component probability:

$$P(S \ \mathbf{AND} \ R) = P(S)P(R|S), \tag{3.64}$$

the reliability of this promised propagation is the probabiliity that there is propagation

$$P(\Delta(R|S)) = P(\pi_+)P(\pi_-|\pi_+) \tag{3.65}$$

where $P(\pi_+)$ is the reliability of keeping the promise π_+. This is a more complicated matter, now involving the join probability distributions for the agents' promises[7]. The basic result is:

$$P(y_- \leq Y \leq y_+|X) = \int_{y_-}^{y_+} \frac{f(x, \alpha)}{f_X(x)} d\alpha \tag{3.66}$$

[7](Need to return to this issue...)

3.6.9 Speed of response propagation

The rate at which influence travels is a distributed quantity, and thus it is subjective

From a god's eye view perspective we note that each promise $\pi^{(\pm)}(A_i)$, made at agent A_i takes time $\Delta T(\pi^{(\pm)}(A_i))$, so that total path time of $A_1, \ldots, A_P$ is:

$$\Delta T_{\text{path}} = \sum_{i=1}^{P} \Big(\Delta T(\pi^{(+)}(A_{i\neq P})) + \Delta T(\pi^{(-)}(A_{i\neq 1})) \Big) \tag{3.67}$$

which in a broady homogeneous system is approximately proportional to the path length, or the total number of promises to be kept.

In the broader scheme of time, the local time taken might vary too, so one can allow some random variation. An initial implementation of state might take longer than the time needed to maintain it.

Some agents experiencing faults may prevent propagation of influence altogether, in which case the total path time tends to infinity, and the local speed of propagation becomes zero.

The time perceived to have passed by any agent in the system depends on the agent's own clock. Each agent will, in principle, measure time differently.

3.6.10 The instantaneous intentional gain

Definition 27 (Intentional gain per agent) *The intentional gain, per agent, in going from a sender set of agents $\{S\}$ to a receiver set $\{R\}$, with a compatible promise binding, may be defined as the number of agents making promises from a sender set. The gain $G_\tau(S,R)$ may be written as the dimensionless normalized value of the response function applied to promise bundle matrices of a type set $\{\tau\}$:*

$$G(S,R,\pi^{\pm}) = \frac{1}{S} \sum_{i=1}^{S+R} \sum_{j=1}^{R+S} \Pi^{+}_{ij} \Pi^{-}_{ji} \left| \frac{\tau(\pi^{+}_{ij}) \cap -\tau(\pi^{-}_{ji})}{\text{sgn}(\tau(\pi^{+}))\tau(\pi^{+})} \right| \left(\frac{\chi(\pi^{+}_{ij}) \cap \tau(\chi^{-}_{ji})}{\chi(\pi^{+}_{ij})} \right) \tag{3.68}$$

Lemma 7 *The maximum gain of any superagent interaction The instantaneous gain of any intentional response is limited to*

$$G(S,R,\pi^{\pm}) \leq |R|, \tag{3.69}$$

i.e. the number of receiver agents.

The proof is follows from the fact that:

$$\left(\frac{\chi(\pi^{+}_{ij}) \cap \tau(\chi^{-}_{ji})}{\chi(\pi^{+}_{ij})} \right) \leq 1, \qquad \frac{1}{S} \sum_{i=1}^{S+R} \sum_{j=1}^{R+S} \Pi^{+}_{ij} \Pi^{-}_{ji} \leq \frac{SR}{S} \to R, \tag{3.70}$$

and

$$\left| \frac{\tau(\pi^{+}_{ij}) \cap -\tau(\pi^{-}_{ji})}{\text{sgn}(\tau(\pi^{+}))\tau(\pi^{+})} \right| \in \{1, 0\}. \tag{3.71}$$

Consider the implications of the lemma. Gain is blind to success or failure. It assumes only that the promise made by the sender is made and kept to some degree. If the promise could not be honoured, and became a bottlneck, e.g. if too many receivers tried to make use of its promise, then the gain would simply amplify the failure to keep the promise by th source. Thus, a failure is not only about whether or not this binding configuration is reliable: if the binding is completely reliable, there is still the possiblity of an error, fault or flaw at source being transmitted perfectly to a larger number of agents, with the same gain.

Definition 28 (Intrinsically amplifying system) *If every agent in a causal dependency process has intentional gain $G > 1$.*

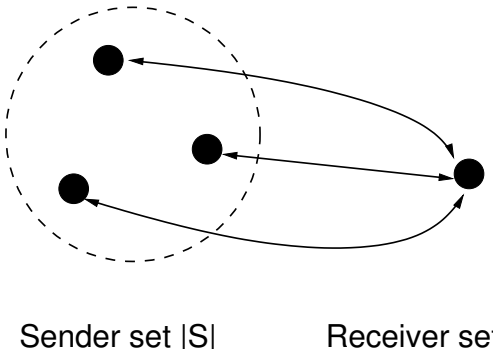

Figure 3.13: A load sharing system is one with unit gain, i.e. the response is exactly the strength of the promise binding.

3.6.11 Impediments to propagation

The category of issues involving reliance on intermediaries or proxies is a large topic. We rely on intermediaries all the time, often without thinking. Indeed, we are so used to ignoring intermediaries for our intent that it becomes a basic source of design flaws: to ignore the promises made by intermediaries. This goes back to our predilection for thinking of the world as an extension of ourselves.

If you are promising to hold on to a mountain, using a rope, you are trusting the rope.

A chain:

$$A \xrightarrow{+c} B \xrightarrow{+b|c} C \tag{3.72}$$

Satellites of agent C:

$$A \xrightarrow{-b} C \xleftarrow{-b} B. \tag{3.73}$$

Same, just no conditional. Both are influenced by C. E.g. C could be a share resource, such as tenancy host. Two tenants are competing for resources. If keeping the promise to one affects C's ability to keep its promise to the other, then there is an undocumented influence between A and B, through this third party. This is called a second order effect.

3.6.12 Propagation of information (awareness)

Awareness of the state of a system is the key to being able to respond quickly to faults, and potentially repairing them before the Nyquist sampling interval expires and the fault has assessed consequences.

The reliability of information, concerning the keeping of promises, moves through a system may be considered a measure of system self-awareness. The information may be destined for a the assessment of a human operator, or for direct assessment and reliance by other component agents. The conundrum for agent networks is that information travelling from one location to another can be distorted by passing through intermediate agents along an information path. In some cases, it might be possible to detect the loss of integrity, but it cannot necessarily be repaired.

3.6.13 Implicit and explicit awareness by repetitive promise keeping

Regular periodic sampling (Nyquist Fourier method) allows us effectively continuous insight into system state of up to half the frequency of the sampling.

State awareness can be basically redundant close to stability, with low frequency random faults, since repeatedly maintaining a convergent state is a stable equilibrium. If we repeat this fast enough, like a heartbeat, we simulate continuity of operational state.

Then we are in some kind of Zen state of knowing without needing to know(!)

3.6.14 Distorted propagation - 'Chinese whispers'

The children's game of Chinese Whispers illustrates how intermediate agents can distort influence, when propagated in a chain. The same scenario was studied with mutual promises rather than impositions, in [BB14], as the Consistent Knowledge Theorem. Let S and R be any two agent nodes, and let $\pi : k$ be a promise body that implies communicating knowledge k. Nodes S and R have consistent knowledge k iff

$$S \xrightarrow{+k_s} R \tag{3.74}$$
$$R \xrightarrow{-k_s} S \tag{3.75}$$
$$R \xrightarrow{+k_r} S \tag{3.76}$$
$$S \xrightarrow{-k_r} R \tag{3.77}$$
$$S \xrightarrow{(k_s=f(k_r,k_s))} R \tag{3.78}$$
$$R \xrightarrow{(k_r=f(k_r,k_s))} S \tag{3.79}$$

where $f(a, b)$ is some function of the original values that must be agreed upon. The proof is given in [BB14].

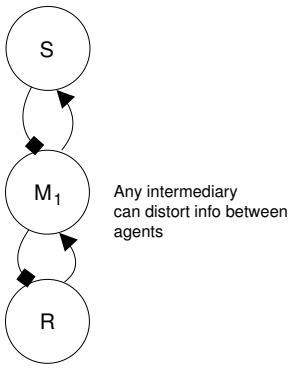

Figure 3.14: Man in the middle and loss of integrity. Shown with impositions.

Knowledge $\mathbf{def}$ can be passed on from agent to agent with integrity, but we must distinguish between agents that end up with a different picture of the information as a result of their local policies for receiving and relaying their knowledge.

1. Accepted from a source, ignored and passed on to a third party intact.

$$\begin{array}{ccc} S & \xrightarrow{+\mathbf{def}_1} & M \\ M & \xrightarrow{-\mathbf{def}_1} & S \\ M & \xrightarrow{\mathbf{def}_1} & R \end{array} \tag{3.80}$$

   Note that the agent M does not assimilate the knowledge here by making its own version $\mathbf{def}_2$ equals to $\mathbf{def}_1$, it merely passes on the value as hearsay.

2. Accepted from a source, ignored and local knowledge is then passed on to a third party instead.

$$\begin{array}{ccc} S & \xrightarrow{\mathbf{def}_1} & M \\ M & \xrightarrow{U(\mathbf{def}_1)} & S \\ M & \xrightarrow{\mathbf{def}_2} & R \end{array} \tag{3.81}$$

   Here the agent M accepts the information but instead of passing it on, passes on its own version. The source does not know that M has not relayed its data with integrity.

3. Accepted and assimilated by an agent before being passed on to a third party with assurances of integrity.

$$\begin{array}{lll} S & \xrightarrow{+\mathbf{def}_1} & M \\ M & \xrightarrow{-\mathbf{def}_1} & S \\ M & \xrightarrow{\mathbf{def}_2=\mathbf{def}_1} & S \\ M & \xrightarrow{\mathbf{def}_2=\mathbf{def}_1} & R \\ M & \xrightarrow{\mathbf{def}_2|\mathbf{def}_1} & R \end{array} \tag{3.82}$$

M uses the data from S, assimilates it ($\mathbf{def}_2 = \mathbf{def}_1$) and promises to pass it on (conditionally $\mathbf{def}_2|\mathbf{def}_1$) if it receives $\mathbf{def}_1$. It also promises to both involved parties to assimilate the knowledge. Only in this case does the knowledge $\mathbf{def}$ become common knowledge if one continues this chain.

The first two situations are indistinguishable by the receiving agents. In the final case the promises to make $\mathbf{def}_1 = \mathbf{def}_2$ provide the information that guarantees consistency of knowledge throughout the scope of this pattern. We can now define scope.

Definition 29 (Scope of common knowledge) *Let S be a set of source agents with consistent knowledge $\mathbf{def}$, and let the set $\mathcal{A}(X, \mathbf{def})$ mean the set of nodes that have assimilated knowledge from a set of agents X. The scope $\mathcal{S}(\mathbf{def})$ of $\mathbf{def}$ is the union of S with all agents that have assimilated knowledge originating from S, i.e.*

$$\mathcal{S}(\mathbf{def}) = S \ \cup \ \mathcal{A}(S \cup \mathcal{S}(\mathbf{def}), \mathbf{def}) \tag{3.83}$$

Comment 19 (Consistency) *The simplest way to achieve common knowledge is to have all agents assimilate the knowledge directly from a single (centralized) source agent. This minimizes the potential uncertainties, and the source itself can be the judge of whether the appropriate promises have been given by all agents mediating in the interaction. This is the common understanding of how a network* directory service *works. Although simplest, the centralized source model is not better than one in which data are passed on epidemically from peer to peer. The problem then is simply in knowing the boundaries of scope. Agents may thus have consistent knowledge from an authoritative source, either with or without centralization.*

3.7 Propagation with branching

3.7.1 Branching with instantaneous serial amplification

The ability to amplify effect is both a useful function and an essential fragility. Amplification of input can also mean amplification of error or fault[8]. A fault at source could at best mean a loss of gain, depending on the promise structure..

Looking at figure 3.15, we see that the the number of source agents $|S| = 4$, and the receiver set has $|R| = 2$ agents.

Definition 30 (Promise valency of an agent) *The number of promises of given type made by an agent, measured in bindings per agent.*

The valency of the agents in the source and receiver sets may be written $s = 2$ and $r = 4$. Because promises of the same type are idempotent[BB14], the binding valency of source agents can never exceed the number of agents in the receiver set, and vice versa, thus:

$$\begin{array}{rcl} r & \leq & |S| \\ s & \leq & |R| \end{array} \tag{3.84}$$

[8]Note that amplification of effect does not depend on whether a system operates by push or by pull, though the susceptiblity for promises not to be kept may do. Any system that has the intent to transfer intent with amplification can propagate and magnify both intended and unintended influence.

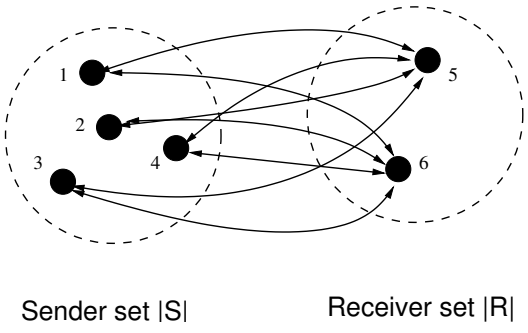

Figure 3.15: Gain in transferring promise bindings from one superagent set $S = 3$ to another receiver set $R = 3$ is proportional to the number of complete bindings and their overlap strength. In this case, if we assume the bindings are equal, the gain would still be 3 times the strength of each binding, since the gain is per agent of the sender set.

The valencies, representing unbound promises, can at best combine in the promise graph to give the product, if each agent binds maximally:

$$\begin{aligned}
\text{Tr}\left(\pi^+(S)\pi^-(R)\right) &= \text{Tr}\left(\begin{array}{cccc|c} & & & & \\ & & 0 & & 0 \\ \hline + & + & + & + & 0 \\ + & + & + & + & \end{array}\right)\left(\begin{array}{cc|cc} & & - & - \\ & & - & - \\ & 0 & - & - \\ & & - & - \\ \hline & 0 & 0 & \end{array}\right) \\
&= \text{Tr}\left(\begin{array}{cccc|cc} 0 & & & & & \\ & 0 & & & & \\ & & 0 & & & \\ & & & 0 & & \\ \hline & & & & r & \\ & & & & & r \end{array}\right) \\
&= rs \qquad (3.85)
\end{aligned}$$

So, we can say that

$$\text{Tr}\left(\pi^+(S)\pi^-(R)\right) \leq rs \leq |R||S|, \qquad (3.86)$$

So

$$\frac{1}{|S|}\text{Tr}\left(\pi^+(S)\pi^-(R)\right) \leq |R| \text{ bindings per source agent.} \qquad (3.87)$$

This is the upper bound for propagation of response. The interesting conclusion here is that amplification doesn't depend on $|S|$ but only on $|R|$. This is slightly counterintuitive, but arises from the spreading of S into R, which is maximized if there are plenty of possible destinations for the source, and we have already measured this relative to the size of the souce pool. It is, of course, possible that some agents do not make or keep the full battery of promises in this kind of redudant arrangement, but it's useful to assume some homogeneity and list a few examples of the amplification.

Example 18 (Gain for some binding configurations) *Assume a cooperative binding between two super-agent clusters (source and receiver) of different sizes, and assume that each agent in these promises to fixed numbers of agents s, r in the other.*

Config	*(+) given*	*(-) accepted*	*Response*	*Gain*	*Comment*
$S \leftrightarrow R$	1	1	$\leq \chi_+$	≤ 1	*Pass through*
$S \leftrightarrow RRR$	1	r	$\leq r\chi$	$\leq r$	*r-linear amplification (multicast)*
$SSSS \leftrightarrow RRR$	s	r	$\leq \frac{rs}{\|S\|}\chi$	$\frac{sR}{\|S\|} \leq R$	*Redundant proxy*
$SSSS \leftrightarrow RRR$	s	$r = 1$	$\leq \chi$	≤ 1	*Redundant switch/load sharing*
$SSSS \leftrightarrow R$	s	1	$\leq \chi$	≤ 1	*Aggregation*

In each case, we can interpret the gain factors as both an amplification of intent and a potential dependency leading to amplification of faults, errors, and flaws in design. Any promise not kept near the source will be amplified by the same functional arrangement.

In multi-stage or multi-tiered cooperative structures, like directed acyclic graphs (DAG) forming branching processes, we can look specifically at the branching of tree-like structures as amplifiers. Staging adds a further dependence on the previous stage, thus some amplification is compounded, which each tier introduces new possibility for failure. The branch points in a tree are called 'single points of failure' (see definition 31), meaning that a fault at one of these has consequences for a fault in the total system. If we add fault-tolerance, the branching simply leads to amplification of the inaccuracy or staleness of the promises, but the system can continue.

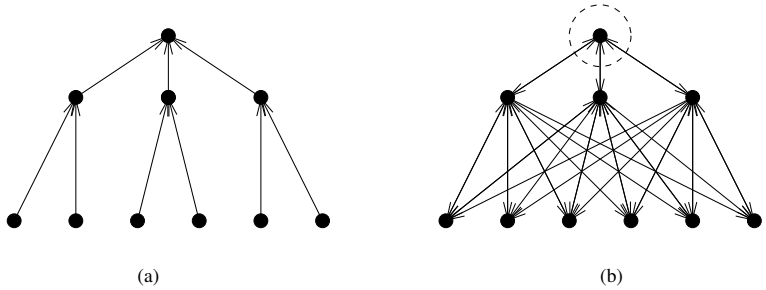

Figure 3.16: Trees are fragile from the root down, as errors are amplified by branching with dependence.

Comment 20 (Human reasoning) *Human reasoning is a branching process that leads to multi-stage instability. Each train of reasoning is non-unique, and multiple viewpoints abound. To assure stability (e.g. through semantic averaging, or selection) one must evaluate all possible trains of thought. This is a time consuming process, hence placing humans in the position of critical decision-making is recipe for instability. The aim must be to decouple reasoning from rapid action, by pre-consideration (modelling). Taxonomy is a model that attempts to exploit branching to avoid inconsistency, unfortunately many properties along which a taxonomy is categorized are shared and non-unique. This often causes anomalies and leads to places where agents should be associated, in a way that contradicts the assumptions of the model.*

3.7.2 Cumulative response

Since the instantaneous response function determines the gain only at a time t, the maximum impact is

$$\text{Impact} = \int_{t_1}^{t_2} G(t)dt \tag{3.88}$$

This applies to fault amplifcation as well as intentional effect. Thus during faults, minimizing the impact can mean minimizing the gain, or minimizing the time over which the fault exists.

- Failover to a redundant component, if possible
- See a repair or replacement (MTTR)

divergence

3.7.3 Branching processes with uncertainty

A branching process is, by its very nature, unstable. To prevent an uncontrolled 'explosion' of effect, we have to contain it to a finite number of agents. Errors may be transmitted or become compounded in series (see fig 3.17).

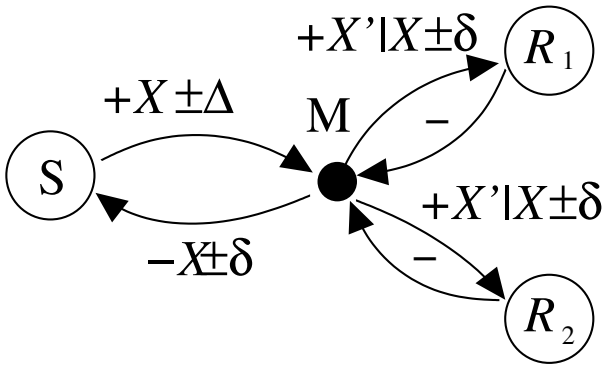

Figure 3.17: An error at the head of a chain of dependences cannot later be mitigated without consequences.

Definition 31 (Single point of failure) *An agent that one or more other agents rely on to fulfil a conditional promise.*

This definition goes beyond a simple physical structure definition of a classical approach to reliability, to include logical promise relationships. A single point of failure is a node that can amplify a fault from a single point to a cascade of derivative promises.

1. S promises service X, within its accuracy tolerance $X \pm \Delta$.
2. Intermediary/relay M, promises to accept the value in the range $X \pm \delta$, where $\max(\delta) < \max(\Delta)$.
3. If Δ happens to be small enough to be within the range $\pm\delta$, it can be accepted and the intermediate agent is able to promise X' to R, within the limits of its own accuracy $\mathrm{err}(\Delta, \delta)$.
4. If $\Delta > \delta$, I is not able to accept the service from X, and cannot promise X' to R, hence there is a complete failure.

If intermediate agents are sufficiently tolerant of inaccurate promise keeping, they can transmit dependent promises within the limits of their own fidelity. However, if the source feeds in a service that is out of the range of tolerance of the intermediary, it must fail to keep its promise, as the condition $S \pm \delta$ is not satisfied

Because of the branching in the latter stage, the error will be amplified by the number of branches, i.e. it will be passed on along each of the causal branches.

3.8 Propagation with convergence

Aggregation, composition, and convergence

The opposite of a divergent amplifying process is a funnel, or convergence from a larger number of agents or states towards a smaller number of agents or states. Aggregation over multiple alternatives may mean redundancy, but not if the aggregated promises are all distinct. In the worst case, aggregation is merely composition, affording no improvement in stability. We have to pay attention to the promise bodies, i.e. the types and constraints, to determine the nature of the aggregation.

3.8.1 Intrinsically converging systems

Definition 32 (Intrinsically converging system) *If every agent in a causal dependency process has intentional gain $G \leq 1$, it may be convergent i.e. it is not amplifying.*

Lemma 8 (Intrinsically converging system with high fidelity) *A strictly converging system with has* $G = 1$ *and* $|R| \leq |S|$

A system is amplifying if $G > 1$, and may therefore converge If $G < 1$, then either the receiver is throttling the or filtering for partial acceptance of the source, or there are faults in propagation of intent. In either case, there is a reduction in effect, whether intentional or unintentional.

3.9 Can we define responsibility for keeping a promise?

Responsibility is an emotionally charged term in system forensics. It can be associated with liability, blame, and reprimand. There is a tendency to want to assign 'blame' a person when faults lead to loss. This is a punitive interpretation of responsibility, and one that does not often make sense, unless willful malice or negligence were demonstrably at work. One also sees attempts to find simplistic root causes (provenance), and quick fixes for future prevention. These tendencies are understandable emotional reactions to loss, but we need a more rational understanding of responsibility, based on the science of causation, to actually stand a chance of making a difference.

Responsibility, in an 'atomic' promise viewpoint, has an interesting simplicity about it. If we can step back from the idea of attributing blame, there is something to be learned from asking the question: which agency or agencies are in a position to be *able* to keep a promise, and hence *could* be considered responsible?

3.9.1 Subjectivity in assessment of faults and errors

The idea of responsibility is diffuse, but it relates to the ability of an agent to respond to a situation in order to keep a promise. A responsible agent (literally a responder) might be one that has decision-making ability, or even full control over whether a promise can be kept (for a stakeholder). In order to be able to make a decision, there have to be alternatives. An agent telling a lie, or exaggerating its claims might be called irresponsible. There is thus an ethical/moral dimension to responsibility too.

The assessment that a promise has not been kept is a subjective one: different players in the system might assess it differently. Their assessments can depend on context or circumstances; so how can we easily attribute a unique source to the perceived failure? This is the challenge of a distributed system with multiple stakeholders.

Example 19 *In a restaurant, a meal is ordered from the menu. One person enjoys the meal, the other doesn't. The latter (dependent agent) assesses the meal to not be what was promised. The former (fault tolerant agent) assesses the meal as acceptable.*

- *The agent that rejects the meal might be considered discerning of quality, but goes hungry and cannot work.*
- *The agent that accepts the meal eats and continues its work.*

Can a cause be attributed to the waiter, the head chef, the sous chef, the butcher?

Whether or not a promise has been kept depends on the kind of promise binding. As we know about promise bindings, the receiver or promisee has to make its own promise to accept what is offered; thus it shares responsibility in outcomes. Indeed, refusing to accept what is offered is the ultimate control decision of autonomous agents.

Example 20 *A doctor promises a patient: if you take these pills you will be cured. If the patient does not keep a promise to take them, then it will not be cured, and thus the responsibility for being cured lies ultimately with the patient, not the doctor.*

3.9.2 Smart and dumb agent responses

A complicating factor in systems is the existence of smart and dumb agents.

Definition 33 (Dumb agent) *A dumb agent keeps its promises but does not adapt to external circumstances.*

Definition 34 (Smart agent) *A smart agent may try to adapt to external circumstances in order to try to keep its perception of responsibility, rather than sticking to the scripted promises.*

When dumb agents fail, they may simply stop or keep trying, but a smart agent might try to adapt. Humans often act as smart agents, but are also asked to forego smart behaviours to remain in a predictable regime. Smart behaviours effectively change the set of promises in real time, in response to external events. This means that the predictability of the promises may be compromised, leading to new behaviours outcomes that were not promised, or expected by others. In this sense, smart behaviours are often frowned upon, because they violate initial expectations. Without criteria for limiting smart behaviours, one must consider intelligent adaptation to be 'out of control'.

3.9.3 The role of conditional promises in pointing to responsibility

In promise theory, we track provenance, or causation with *conditional promises*. Each promise is the responsibility of the agent who makes the promise (the promiser). From the conditional promise law, an agent making a conditional promise has not made a promise at all unless it also promises to acquire the thing its promise is conditioned on.

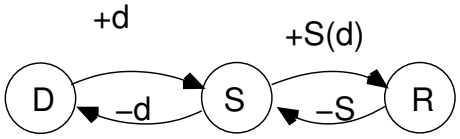

Figure 3.18: A conditional promise chain, showing service delivery based on a dependency.

Consider the scenario in figure 3.18

$$D \xrightarrow{+d} S \tag{3.89}$$
$$S \xrightarrow{-d} D \tag{3.90}$$
$$S \xrightarrow{+S(d)} R \tag{3.91}$$
$$R \xrightarrow{-d} S \tag{3.92}$$

where

$$S \xrightarrow{+S(d)} R \equiv \begin{cases} S \xrightarrow{+S|d} R \\ S \xrightarrow{-d} R \end{cases} \tag{3.93}$$

This system is fragile because the recipient has only a single choice. It has a single point of failure. If The recipient could seek out redundant alternatives to provide the service S.

What happens beyond the horizon of the next agent in the chain of promise relationships is beyond the control of the recipient, and is thus beyond the limit any possible responsibility.

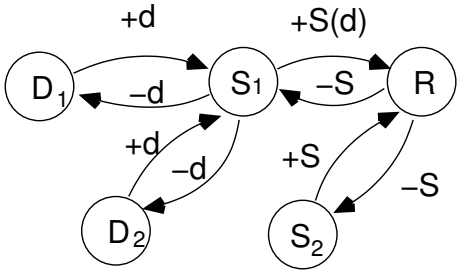

Figure 3.19: A redundant conditional promise chain, showing service delivery based on a dependency.

Now consider the same scenario with redundancy along the chain (see figure 3.19)

$$D_1 \xrightarrow{+d} S_1 \tag{3.94}$$
$$D_2 \xrightarrow{+d} S_1 \tag{3.95}$$
$$S_1 \xrightarrow{-d} D \tag{3.96}$$
$$S_1 \xrightarrow{+S|d} R \tag{3.97}$$
$$S_1 \xrightarrow{-d} R \tag{3.98}$$
$$S_2 \xrightarrow{+S} R \tag{3.99}$$
$$O \xrightarrow{-d} \{S_1, S_2\} \tag{3.100}$$

In this second scenario, both the server S_1 the recipient can choose from two providers of the promises they are trying to use. For the final recipient R, the fact that the promise from S_1 has a dependency is irrelevant, as there is nothing it can do about that except to acquire a second provider who may or may not have a dependency too. The only security the recipient R has is to have a choice of providers. No matter how hard the providers S_1 and S_2 try to keep their promises of service, unforeseen circumstances may prevent them from doing so. Indeed R may itself be negligent receiving their services.

This suggests that, while reponsiblity for keeping a promise lies with each source agent, only the final recipient can be considered responsible for securing a successful promise outcome.

3.9.4 A downstream principle for responsibility: locality

Locality gives a surprisingly simple and consistent interpretation of responsibility. The recipient of a promise carries the burden of outcome.

In a chain of promises, dependencies are upstream (the source of the flow of influence) and the benefactors are downstream. Based on the consistency of responsibility described above, we can make the following straightforward observation. The assurance of the final promise outcome follows a 'downstream principal' that the most downstream agent has both access and opportunity to correct or absorb faults, and hence the greatest causal responsibility for an assessment of a promise not being kept. In other words, the greater the distance from the point of promise-making, the less causal responsibility an agent has in contributing to the outcome.

!!!!!!!!!!DIAGRAM!!!!!!!!!!

This is not a moral assessment, it is a purely pragmatic observation about cause and effect. However, it is interesting that it is in opposition to what is conventionally assumed about fault-tree hierarchies and root cause analysis. The explanation for this apparent contradiction can be found in the bi-directionality of promise bindings required for propagation of influence. The traditional assumption has been that influence is always 'pushed' and that pushes always succeed, the latter being false (see section 3.5).

Can we assert, then, that an agent who fails to use a promised service in order to keep its own promise is more responsible than the failure of the agent to provide the service? The user of the service could, in principle, seek a redundant alternative for such cases. But what if no alternative is available?

If a promise is made conditionally, the agent (promiser) advertises a delegation of its responsibility to keep the outcome, by adding conditions. The promiser of each link in a chain of cooperation is responsible only for its own promises, i.e. the point at which it can effect change of behaviour or intent. This tells us that delocalization means divesting responsiblity to others. This is the power of agent autonomy. Thus, in promise theory, reponsibility is connected to locality. How does this compare to conventional component reliability theory?

- In the black box tradition of reliability theory, there is insufficient information to be able to attribute provenance for observed failures, so one can only attribute failure to a component itself. The failure of a component must be considered a random event, and the lack of information means that it makes no sense to assign moral blame.

- In promise theory, the assumption is that an agent that makes a promise is the only agency responsible for keeping the promise. It might be possible to argue moral or causal blame to the specifically documented promise. This is in keeping with the tradition of legal responsibility.

- By the conditional promise law, a promise that is conditional on another promise being kept (either by the same agent or by a third party) is not a promise, unless the other promise is made by the same agent. This clarifies and documents diminished responsibility.

We can now attempt a limited but tenable definition of responsibility:

Definition 35 (Responsibility for keeping a promise) *Responsibilty has two main interpretations:*

- Causal responsibility *is a probability distribution for the keeping of a promise outcome, described over a collection of agencies, that attribute the relative influence of agents to the outcome that promise. It is an assessment of whether a fault is result of the agent itself or of a prior dependency. This can only be attempted post-hoc, and is situation specific. It is a systemic issue.*
- Moral responsibility *(culpability) is a human assessment, about whether agent outcomes stem from good or for bad intent, hence it cannot be formalized except as a norm or in law. This is not a systemic issue.*

Comment 21 (Causal and moral responsibility) *An agent cannot be solely responsible for keeping a conditional promise. Could an agent that relies on a promise from another be culpable for a failure to find an alternative if the dependency fails to keep a promise upon which it relies? This would be a case of negligence.*

Chapter 4

Scaling system promises

So far, we've looked at small constellations of agents with only a few promises at a time. If agents are thought of like atoms, then we could call these molecular systems: clusters of atoms that bring about new functional behaviour. In many cases, this is helpful already for deriving some insight into the workings of systems. However, the real benefits of formalization come when we look at greater scales.

Scaling is one of the techniques that physicists have used to address the generic patterns and broad economics of systems, both measured with energy and with money. The general rule, in a wide class of systems, is that the more you scale up, the more universal (and hence less specific) the results become. Sometimes one can characterize 'efficiencies of scale', in which costs scale more slowly than outputs.

Information science does not usually think of scaling in this way. Rather, it talk about 'scalability', or the use of parallelism to increase flow by increasing the size of a supporting system infrastructure. In other words, it concerns itself mainly with increasing dynamical throughput. This is is purely dynmaical consideration; but, information rich systems, including but not limited to software, are dominated by the *semantics* of their interactions, so there is a pressing need to understand the scaling of semantics too[Bur14, Bur15]. There are thus two meanings of scaling:

1. **Type 1: scaling of workload (parallel support)**.

   Scaling up a system refers to the intended increase in size or load bearing characteristics by increasing the number of agents, or by altering their promises. We compare systems based on whether they can keep a promise independently of the physical dimensions of the system, or the magnitude of the load.

2. **Type 2: scaling of agency (spacetime bulk)**.

   This scaling refers to a change in the way we probe or interact with a system, either in detail (small scale, short wavelength) or in bulk (large scale, long wavelength), and the effect that it has on what bulk promises are kept. This is sometimes used to make scaled models.

In 1. we are asking whether a promise can be kept for a larger load of interacting agents that still interact with the parts at the same scale according to 2. In 2. we are asking how big are the effective agents we are examining.

Definition 36 (Functional scalability) *A system possesses this property if it can change its size (number of agents) without altering its function, or changing its ability to keep promises (as observed by an external agent).*

When an architect or designer makes a scaled model, he or she is using the idea that by magnifying all of the dimensions of a building or device in proportion, there will be a predictable correspondance between the behaviour of the model and the full-size result. This is not a trivial matter; it presupposes a notion of dynamical similarity[Bur13, Bar96, Bar03].

4.1 Lessons from effective coarse descriptions

The more we understand how systems work, the more chance we have of forming the right expectations about them, and using them successfully for their semantics as tools. Scales thus play a unique role in understanding how systems behave.

When we are unable to separate the characteristics of a system at different scales, we fail to understand the semantics of the system, and see only the interactions of its parts. The phrase 'can't see the wood for the trees' expresses this: if all you see is trees, you can't perceive a forest or wood as an identifiable, aggregate entity, or know what role it plays in climate interactions, or global biodiversity, etc.

At each level of aggregate description, there are typically new patterns and emergent characteristics that cannot be understood from the individual parts, in the absence of their interactions. There can be new semantics associated with these (such as when a collection of components becomes a computer). In promise theory, we call these aggregations superagents. At coarser scale, dynamics become more 'universal' in their characters, i.e. we see what is common to all forests rather than what is special about one.

4.1.1 Ensemble scaling and universality of characters

System characteristics can be assessed statistically over 'ensembles' or collections of comparable systems. This is particularly useful when faced with real-world indeterminism. Sometimes we form ensembles over time, to discuss trends of change; other times, ensembles may be picked from different locations and circumstances to compare locations at the same time[1].

Some of these characteristics are famous, like Moore's law, which characterizes the development of a state by saying that the number of transistors on chips doubles every two years. This is dynamical law of scale, often interpreted semantically to mean that computing power or performance doubles every two years, which is really a different question. Similarly, there is Wirth's law[Wir95] which suggests that software is actually getting slower more rapidly than hardware becomes faster. The semantic association here is one of increasing bloat and complexity in software. Another example is how we tend to be rosy eyed about places and things we don't know very well. If you live in a place for a long time, you probe it in greater detail, and experience more of its problems. This could be formalized into a law: the longer you live somewhere, the more you complain about it.

Formal scaling relations are rarely found in information technology, because they are the more familiar tool of physicists: the physics of information systems is a relatively new area of study, motivated partly by the growth of systems to respectable scales. Recently studies exposing universal scaling in cities were carried out[Bet13, BLH$^+$07], revealing how key performance indicators like wages, disease, services, etc., grow according to universal patterns. Cities are semantically rich information systems, as diverse as any technology. This suggests that high level patterns are universal, as we would expect on general scaling principles, and that they are likely to exist in other functional systems.

Some scaling relations expose patterns across ensembles of systems that are parameterized by size, weight, length, or some measurable dimension. Scaling is connected with dimensionless variables, such as numerical counts N. When control parameters have dimensions (mass, length, time, etc.) the control parameters are dimensionless ratios: t/T, m/μ, and so on.

Scaling relations can be compared with great generality across many systems, and correspondingly lead to only general observations that don't necessarily describe any of the samples in an ensemble well. So, while this is scientifically interesting, does any of it help us to design and build systems, or predict faults, flaws or even errors? How might the semantics at aggregate scales, and even universal scaling relations, be disrupted in a way that became anomalous?

4.1.2 Time

Time and space (size) are intertwined in sometimes unexpected ways. In an intentional system, time does not only affect dynamics, but also semantics, because observers perceive and interpret behaviour according to their own notions of time.

The speed of propagation of influence, i.e. communication, is what connects distance and time together. Processes measure time, and interactions, which mix scales through space, also lead to the emergence of multiple scales in time: as systems become larger in space, they often become slower in bulk. Since time is an important part of whether a promise is considered kept, scaling can lead to a new subtlety. Large systems might simply fail

[1] The meaning of 'at the same time' is itself ambiguous. Simultaneity in a system is one of the difficult concepts in relativity, as it assumes a global definition of time, which might not be possible.

to keep their short term promises. Conversely, the accumulation of interactions from a larger system may place burdens on resources, so that what works for the short time, might not work in the long run.

Beyond these fairly obvious examples, there may be complicated spacetime involvement in which the geometry of a system funnels work (through dynamical constraints) in a particular way, and a system needs new policy tradeoffs (i.e. new semantics) to handle the new behaviours. It is a reasonable hypothesis that such changes in scale, whether intentional or unintentional, are responsible for many unexpected 'failures' in systems.

4.1.3 Separation of scales

When we try to scale systems, intentionally, to encourage greater output (such as building a larger factory, or a bigger datacentre, adding more servers, etc), we risk the fact that system mechanisms will not actually scale in accordance with our intentions. Designing systems for a large scale, or to span several scales, warrants different considerations compared to those for a small ad hoc system.

Definition 37 (Coarse graining) *The elimination of detail from a system by aggregating low level details into fewer variables that represent the same situation over a new aggregate scale.*

Statistical averaging is one coarse graining procedure. After coarse graining over space, the *interior* details of a region are no longer observable, because we probe the system only as the average effects over longer distances. A consequence of this is that we are free to focus on the effective *exterior* interactions at the new scale, between the grains.

If we take an ensemble of systems, of different sizes, in similar circumstances, which keep a specific set of *exterior* promises, they might fail in different ways because of unobservable differences in the networks of interior promises. In other words, grains might appear to be superficially similar, whereas in fact they are different. The microscopic specifics of an system may play a role, if they remain significant as we scale.

Principle 1 (Separation of scales) *In a weakly coupled (quasi linear) system, the details at one scale do not lead to strong effects at a different scale. This principle breaks down in non-linear systems with stroing coupling.*

Non-linearity in systems is a source of great unpredictabilty because it prevents decoupling of scales. Small details can be amplified into large scale effects through interactions.

4.1.4 Scaling semantics: system function

Addressing semantics on the level of atomic and molecular combinations of agents is straightforward, if not intricate enough, but how could we describe systems of such great size and intricacy that it is impractical to look at every promise individually? This has been studied using promise theory in [Bur14, Bur15].

We can learn from material science, i.e. how to describe systems without tracking every atom in a substance. Phenomena manifest at different scales with different expressions. By eliminating detail and moving to an effective coarse description, we might lose total information, but the lesson from the physics is that this doesn't necessarily matter. The laws of scaling and 'renormalization' can keep track of how small details either lose their importance at scale, or get amplified into chaos.

For example, we can say that certain crystalline structures are susceptible to cracks. Some are flexible and resilient. More practically, we can say things like: don't build a hammer out of glass as it is unsuitable. Such properties are usually learned from experience and research rather than being a priori predicted from models, though today the information technology does allow predictions. Elucidating scaled properties of abstract 'human-machine materials' is a place to start, even if we can't immediately see all of the motivations and consequences: we follow the example of material science, hoping for enlightenment down the road.

4.1.5 Scaling design flaws: what semantics can change?

Design issues that are sensitive to scaling include costs and benefits in a number of areas. Some issues are controllable (they are short range effects). Others are dominated by the environment and thus beyond control (they are long range effects).

- Maintenance.
- Replacability.
- Transport of people, materials and information.
- Latency.
- The efficiency of resource supply, harvesting of output, and maintenance come into play.

One of the things a larger system can afford is to be more tolerant of faults; this may come from having sufficient redundancy of dynamical processes to sustain functional behaviour. However, when the fault lies in an external dependence, its own redundancy may not help. So a multiply redundant flight control computer on an aircraft is no help if the plane runs out of fuel.

Offering

- Backups, dynamical multitude, redundant agents keeping the same promise for failover.
- Alternatives, semantic mulitude or different promises for a different solution.

These are two ways of building flexibility and resilience into systems. They can be combined with 'mixing' or random selection (so that agents do not always bind to the same promise provider). In that way, a fault in a single location can be mitigated by varying the selected location. This introduces a new scale, e.g. the timescales for the bindings.

4.2 Scaled agents (sub- and super-agency)

The treatment of a collection of agents as a single entity is a choice of scale, made by any observer (see figure 4.1). Agency can further be defined recursively to build up hierarchies of component parts, and their dependencies.

Definition 38 (Long and short range coupling) *If dependencies are entirely self-contained, there is a natural local scaling. If dependencies remain between agents, even at a larger coarser scale, then there are long range effects.*

In [Bur14], I showed how spatial boundaries can be defined by membership to a group or role. We still have to explicate the relationship between the internal members and the structure of the whole, as perceived by an observer.

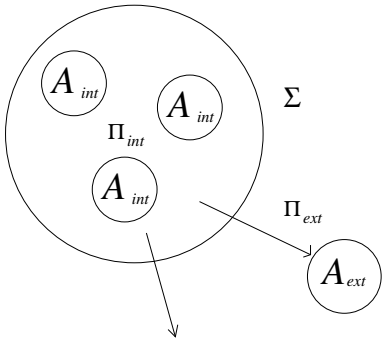

Figure 4.1: Agent structure consists of an element that makes a number of exterior promises, some of which are scalar, some vector, etc. Interior promises are invisible from the outside. For example, the subagents might promise to be trees (and not suddenly become cars), while the superagent promises to be a wood.

We define a collective super-agent as a spacetime structure that has collective agency, i.e. its intended semantics relate to a collection of agents surrounded by a logical boundary, with collective semantics (see figure 4.1).

Definition 39 (superagent) *A superagent of size S is any bounded agency composed of individually separable agencies, partially or completely linked by internal vector promises. The bare superagent is defined by the closed graph, without any external adjacencies.*

Super-agency allows us to talk about redundant 'material' bulk promises. In principle an observer could draw a line around any collection of agents and call it a cell or composite superagent. This is an assessment any agent can make, as part of its definition of an agency scale. However, it might still be of interest to distinguish special criteria by which such an arbitration might occur. In component design, for instance, the choice of boundary has often to do with the a choice interface an agent wants to interact with.

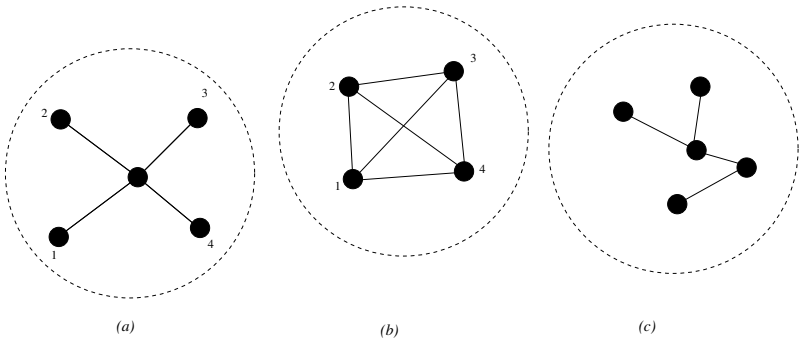

Figure 4.2: Three ways of binding collective agency. Another way is to simply make an arbitrary collection: (a) membership, (b) interaction mesh, (c) chain or dependency hierarchy.

The alternatives fall into three basic categories (see figure 4.2):

(a) A membership in a group or associative role, where the central membership authority may be either inside or outside the boundary, e.g. city limits, or company campus. In this case, we are identifying a group of symmetrical agents.

$$A_{\text{host}} \xrightarrow{+\text{membership}} \{A_{\text{tenant}}\} \tag{4.1}$$

$$\{A_{\text{tenant}}\} \xrightarrow{-\text{membership}} A_{\text{host}} \tag{4.2}$$

(b) A total graph or collaborative role. In this case, we are identifying agents with coordinated behaviours.

(c) A dependency graph, path or story. In this case we are identifying dependency bindings.

If we have superagents, then obviously we can talk a bout subagents, where a particular system region can be decomposed into helpful parts. Lesser agents can also be residents or satellites of other agents. This leads to a hierarchy of containment by functional role.

4.2.1 Superagent surface boundary

Superagent boundaries may be formed with different structural biases.

- Aggregations of agents, related through membership or allegiance to a single leader (leader may be inside or outside the superagent), and the connections are made through the leader as a proxy-hub.
- A cluster of agents linked by cooperative vector promises.
- Strongly cooperative agents which are inseparable without breaking an external promise. E.g. an organism made of components that are all different and non-redundant to the functioning of the whole.

Interior promises are those entirely within the surface boundary of a superagent.

Definition 40 (Surface of a superagent) *The exposed* surface Σ *of the agent is the subset of interior/internal agents that have adjacencies to agencies outside the superagent.*

A superagent surface may also make new explicit promises that are not identifiable with a single component agency.

4.3 Type 1 (workflow) scaling with added workforce

Throughput scaling, at constant semantics, is about characterizing the output of a system as a function of input (usually a task). The ability of a system to complete tasks depends on the extent to which the task can be broken up into independent streams, which can be completed in parallel, and recombined into a coherent whole. Thus the topology of the system and its communication channels play a role. Some simple ideas about scalability can be understood using the flow approximation of queueing systems, in which a promise kept involves a continuous stream of data, observations, changes, etc.

4.3.1 Current/flow approximation

We can think of scalability as the ability of a system to deal with flows of input, or more correctly, we consider how increasing the input affects the efficiency with which tasks are discharged at the output. This sounds somewhat like the problem posed in queueing theory. However, whereas queueing theory deals with the subtleties of random processes, scalability is usually discussed in a pseudo-deterministic flow approximation.

A set of models, defined in [BC04], described the one dimensional scaling configurations, and costs, associated with maintaining promises in order of decreasing centralization, or increasing delegation. One could imagine that this ordering also corresponds, roughly, to decreasing level of determinism about the network. However this interpretation may be misleading, since even centralized control schemes are prone to noise, and local or even catastrophic system-wide failure.

The cases are presented in table 4.1 below, and the authors assumed a simple linear relationship between the probability of a promise being kept and the rate(s) of communication with the policy- and enforcement-source(s).

In promise language, a promise-keeping verification ('charge') ΔQ is proportional to an average rate of assessment (information flow or current) I, over a time Δt; that is $\Delta Q = I\Delta t$. This equation is valid when I represents the time-averaged flow over the interval. Since we are interested in the limiting behaviour for long times, this is sufficient for our needs.

Now we apply this simple picture to promise keeping in dynamic networks. We take the point of view of a 'typical' or 'average' host. It fails to keep its promise at the (average) rate $I_{\rm fault}$, and receives corrections at the rate $I_{\rm repair}$. Hence the rate of increase of faults for the average node is:

$$I_{\rm fail} = (I_{\rm fault} - I_{\rm repair})\,\theta(I_{\rm fault} - I_{\rm repair}). \tag{4.3}$$

The Heaviside step-function is defined by $\theta(x) = 1$ if $x > 0$ and $\theta(x) = 0$ if $x <= 0$, and signifies the fact that, if the repair rate exceeds the faults rate, then (on average, over long times) nothing remains outstanding and there is no net rise in configuration faults. Thus this averaged quantity is never negative.

If random faults and changes to configuration occur at a rate $I_{\rm fault}$ and the configuration agent is unavailable to correct them, then $I_{\rm fail} = I_{\rm fault}$. If this holds during a time Δt, the configuration falls behind by an amount:

$$\begin{matrix}\text{Information}\\ \text{missing}\\ (\Delta Q)\end{matrix} = \begin{matrix}\text{information/sec}\\ (I_{\rm fault})\end{matrix} \times \begin{matrix}\text{seconds}\\ \text{unavailable}\\ (\Delta t)\end{matrix} .$$

In the following we will use p to denote the average (over time, and over all nodes) probability that configuration management information flow (repair current) is not available to a node. This unavailability may come from either link or node unreliability. We can lump all the unreliability into the links (see above) and so write $p = (1 - \langle A_{ij}\rangle)$, where $\langle A_{ij}\rangle$ denotes both time and node-pair average. Each node then can only receive repair current during the fraction $(1 - p)$ of the total elapsed time.

The repair current is generated by two possible sources in our models: i) a remote source, and ii) a local source. In each case, the policy can be transmitted and/or enforced at a maximum rate given by the channel capacity of the source. We shall denote the channel capacities by C_R and C_L for remote and local sources for clarity, but we assume that $C_R \sim C_L$, since source and target machines are often comparable, if not identical. If the communication by network acts as a throttle on these rates, then one can further assume that $C_R < C_L$. In any case, the weakest link determines the effective channel capacity. Note that in the case of a confluence of traffic, as in the star models below, the channel capacity will have to be shared by the incoming branches. We now have a criterion for eventual failure of a promise strategy. If $I_{\rm fail} = \frac{\Delta Q}{\Delta t} > 0$, the average fault current will grow

Table 4.1: Comparison of models from the viewpoint of the different dimensions: policy dissemination, enforcement, freedom of choice, whether hosts can exchange chosen policy ideas with peers and how political control flows. A 'push' model implies a forcible control policy, whereas 'pull' signifies the possibility to choose. Model 3 lies between these two, in having the possibility but not the inclination to choose.

Model	Application Topology	Enforcement	Policy Freedom	Policy Exchange	Control Structure
1	Star	Remote imposition	No	No	Radial push
2	Star	Remote imposition	No	No	Radial push
3	Mesh	Local promise	No	No	Radial pull
4	Mesh	Local promise	Yes	No	Radial pull
5	Mesh	Local promise	Yes	Yes	Hierarchical pull
6	Mesh	Local promise	Yes	Yes	P2P pull

monotonically for all time, and the system will eventually fail in continuous operation. Our strategy is then to look at the scaling behaviour of I_{fail} as the number of nodes N grows large.

Star model (deterministic)

The starting point for our models is the traditional (idealized) model of host configuration, based on the idea of remote impositional management. Here there is a central manager who decides and implements policy from a single location, sending instructions that read and write necessary state. In this first case, all of the network connections and hosts are considered to be completely reliable. The manager must supply or watch the whole network, using bi-directional communication. This leads to an $N:1$ ratio of clients to manager (see fig 4.3). This

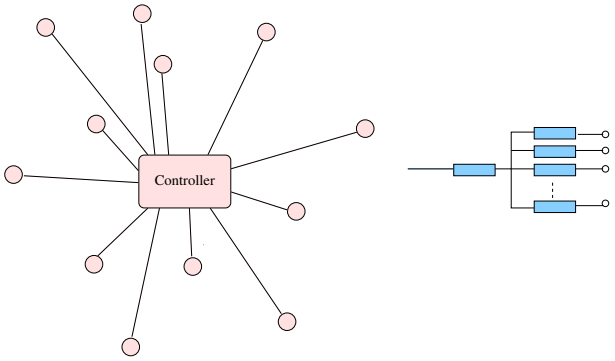

Figure 4.3: Model 1: the star network. A central manager maintains bi-directional communication with all clients. The links are perfectly reliable, and all enforcement responsibility lies with the central controller.

first model is an idealized case in which there is no unreliability in any component of the system. It serves as a point of reference.

The topology on the left hand side of fig 4.3 is equivalent to the component topology on the right hand side. We can assume a flow conservation of messages on average, since any dropped packets can be absorbed into the probabilities for success that we attribute to the adjacency matrix. Thus the currents must obey Kirchoff's law of conservation:

$$I_{\text{controller}} = I_1 + I_2 + \ldots I_N, \tag{4.4}$$

where the currents are the assessed outcomes of impositions:

$$I_n = \alpha_{\text{Controller}} \left(\text{Controller} \xrightarrow{+I_n} \text{Client}_n \right). \tag{4.5}$$

The controller current cannot exceed its capacity, which we denote by C_S. We assume that the controller puts out repair current at its full capacity (since the Heaviside function corrects for lower demand), and that all nodes are

average nodes. This gives that

$$I_{\rm repair} = \frac{C_S}{N}. \tag{4.6}$$

The total current is limited only by the bottleneck of queued messages at the controller, thus the throughput per node is only $1/N$ of the total capacity. We can now write down the failure rate in a straightforward manner:

$$I_{\rm fail} = \left(I_{\rm fault} - \frac{C_S}{N} \right) \theta \left(I_{\rm fault} - \frac{C_S}{N} \right). \tag{4.7}$$

In other words, the failure rate is zero until the total rate of faults exceeds the capacity of the channels, at which point it scales inversely with N.

As the number of hosts becomes very large, as in a pervasive environment, $N \to \infty$, $I_{\rm fail} \to I_{\rm fault}$—i.e., the controller contributes a vanishing repair current per node and no faults are corrected. At this point, the system is out of control. The system fails however at a finite $N = N_{\rm thresh} = C_S/I_{\rm fault}$. This highlights the clear disadvantage of centralized control, namely the bottleneck in communication with the controller.

Star model in intermittently connected environment

The previous model was an idealization, and was mainly of interest for its simplicity. Realistic centralized dependence must take into account the unreliability of the promise keeping environment.

In an environment with partially reliable links, a remote communication model bears the risk of not reaching every host. If hosts hear policy, they must accept and comply, if not, they fall behind in the schedule of configuration. Monitoring in distributed systems has been discussed in ref. [ALB99]. The capacity of the central manager

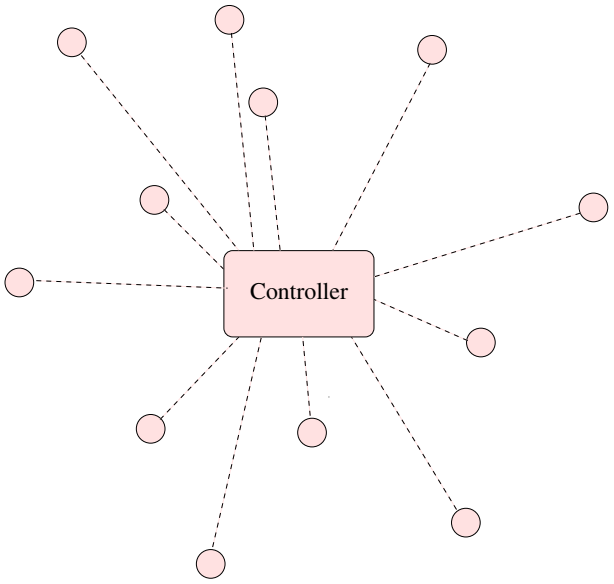

Figure 4.4: Model 2: a star model, with built-in unreliability. Enforcement is central as in Model 1.

C_S is now shared between the average number of hosts $\langle N \rangle$ that is available, thus

$$I_{\rm repair} = \frac{C_S}{N \langle A_{ij} \rangle} \equiv \frac{C_S}{\langle N \rangle}, \tag{4.8}$$

where $\langle N \rangle \leq N$. This repair current can reach the host, and serve to decrease its policy faults ΔQ, during the fraction of time $(1-p)$ that the typical host is reachable. Hence we look at the net deficit ΔQ accrued over one 'cycle' of time Δt, with no repair current for $p\Delta t$, and a maximal current $C_S/\langle N \rangle$ for a time $(1-p)\Delta t$. This deficit is then

$$\Delta Q(\Delta t) \quad = \quad I_{\rm fault} p \Delta t + \left(I_{\rm fault} - \frac{C_S}{\langle N \rangle} \right) (1-p) \Delta t \tag{4.9}$$

(here it is implicit that a negative ΔQ will be set to zero). Thus, the average failure rate is

$$I_{\text{fail}} \quad = \quad I_{\text{fault}} p + \left(I_{\text{fault}} - \frac{C_S}{\langle N \rangle} \right) (1-p) = I_{\text{fault}} - \frac{C_S}{N}. \tag{4.10}$$

(Again there is an implicit θ function to keep the long-time average failure current positive.) This result is the same as for Model 1, the completely reliable star. This is because we assumed the controller was clever enough to find (with negligible overhead) those hosts that are available at any given time, and so to only attempt to communicate with them. If $\langle N \rangle \ll N$, there is potentially room for the controller to handle a much larger N with the same capacity.

This model then fails (perhaps surprisingly), on average, at the same threshold value for N as does Model 1. If the hunt for available nodes places a non-negligible burden on the controller capacity, then it fails at a lower threshold. For large N, we have:

$$I_{\text{fail}} \to I_{\text{fault}}. \tag{4.11}$$

Note that we now have an interesting conclusion. It can actually pay, for some N to have certain agents down or unavailable, since this will increase the throughput to the central manager and delay the inevitable failure.

Mesh topology with centralized policy and local enforcement

The serialization of tasks in the previous models forces configuration 'requests' to queue up on the central controller. Rather than enforcing policy by issuing every instruction from the central source, it makes sense to download and cache a summary of the policy to each host and empower the host itself to enforce it.

There is still a centrally determined policy for every agent, but now each host carries the responsibility of configuring itself. There are thus two issues: i) the update of the policy and ii) the enforcement of the policy. A pull model for updating policy is advantageous here, because every host then has the option to obtain updates at a time convenient to itself, avoiding confluence contentions; moreover, if it fails to obtain the update, it can retry until it succeeds. We ask policy to contain a self-referential rule for updating itself.

The distinction made here between communication and enforcement is important, because it implies distinct types of failure, and two distinct failure metrics: i) distance of the locally understood policy from the latest version, and ii) distance of host configuration from the ideal policy configuration. In other words: i) communication failure, and ii) enforcement failure. The host no longer has to share any bandwidth with its peers, unless it is updating its

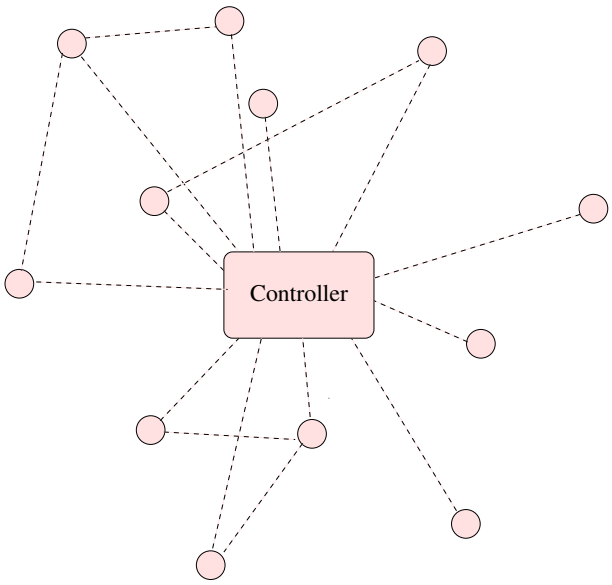

Figure 4.5: Model 3. Mesh topology. Nodes can learn the centrally-mandated policy from other nodes as well as from the controller. Since the mesh topology does not assure direct connection to the controller, each node is responsible for its own policy enforcement.

copy of the policy, and perhaps not even then, since policy is enforced locally and updates can be scheduled to avoid contention.

Let $I_{\rm update}$ be the rate at which policy must be updated. This current is usually quite small compared to $I_{\rm fault}$, and was neglected in the previous models. Based on the two failure mechanisms present here, we break up the failure current into two pieces: $I_{\rm fail} = I_{\rm fail}(i) + I_{\rm fail}(ii)$. The former term is

$$I_{\rm fail}(i) \quad = \quad (I_{\rm fault} - C_L)\theta(I_{\rm fault} - C_L)\,; \tag{4.12}$$

this term is independent of N and may be made zero by design. $I_{\rm fail}(ii)$ is still determined by the ability of the controller to convey policy information to the hosts. However, the load on the controller is much smaller since $I_{\rm update} \ll I_{\rm fault}$. Also, the topology is a mesh topology. In this case the nodes can cooperate in diffusing policy updates, via flooding[2].

The worst case—in which the hosts compete for bandwidth, and do not use flooding over the mesh—is that, for large N, $I_{\rm fail} \to I_{\rm update}$. This is a great improvement over the two previous models, since $I_{\rm update} \ll I_{\rm fault}$. However note that this can be further improved upon by allowing flooding of updates: the authorized policy instruction can be available from any number of redundant sources, even though the copies originate from a central location. In this case, the model truly scales without limit, i.e. $I_{\rm fail} = 0$.

Comment 22 (Ad hoc networks) *There is one caveat to this encouraging result. If the (meshed) network of hosts is truly an ad-hoc network of mobile nodes, employing wireless links, then connections are not feasible beyond a given physical range r. In other words, there are no long-range links: no links whose range can grow with the size of the network. As a result of this, if the AHN grows large (at fixed node density), the path length (in hops) between any node and the controller scales as a constant times $\sqrt{N}$. This growth in path length limits the effective throughput capacity between node and controller, in a way analogous to the internode capacity. The latter scales as $1/\sqrt{N}$ [GK00, LNP90]. Hence, for sufficiently large N, the controller and AHN will fail collectively to convey updates to the net. This failure will occur at a threshold value defined by*

$$I_{\rm fail}(ii) = I_{\rm update} - \frac{C_S}{c\sqrt{N_{\rm thresh}}} = 0, \tag{4.13}$$

where c is a constant. The maximal network size $N_{\rm thresh}$ is in this case proportional to $\left(\frac{C_S}{I_{\rm update}}\right)^2$—still considerably larger than for Models 1 and 2.

An alternative to flooding is to simply rely on the central server for updates, as in Models 1 and 2. In this case, the update term $I_{\rm fail}(ii)$ is the same as for those models, with $I_{\rm fault}$ replaced by $I_{\rm update}$. This is worse than the flooding method above.

Mesh topology with partial host autonomy and local enforcement

As a variation on the previous model, we can begin to take seriously the idea of distance from a controlling centre. In this model, hosts can choose not to receive policy from a central authority, if it conflicts with local interests.

Example 21 *Embedded ad hoc systems are likely to involve many personal devices, goverened by their owners rather than promising to comply with any standard. These devices will roam, like tourists, through regions of foreign policy where they must either agree of conform to local rules or defy them. In such an environment, personal freedoms come into play and users will demand the freedom to decide over their own property. No absolute control will be acceptable.*

Communication thus takes the role of conveying 'suggestions' as a service from a local central authority, in the form of the latest version of its policy. Policy works now by voluntary cooperation. For instance, a central authority might insist on the use of a particular vendor's proprietary product in order to qualify for local service provision. This might be unacceptable to the individual client who would prefer to defer its access to the service until a more acceptable solution is available. In another scenario the central authority might suggest a new version

[2]Note, flooding in the low-level sense of a datagram multicast is not necessarily required, but the effective dissemination of the policy around the network is an application layer flood.

of widely-used software, but the the local authority might delay the upgrade due to compatibility problems with local hardware.

To implement this kind of freedom, local enforcement is now a necessity. Each node is empowered to hold to its chosen policy P_i. Thus communication and enforcement use formally distinct channels (as with Model 3); the difference is that each node has its own target policy P_i which it must enforce. The communications and

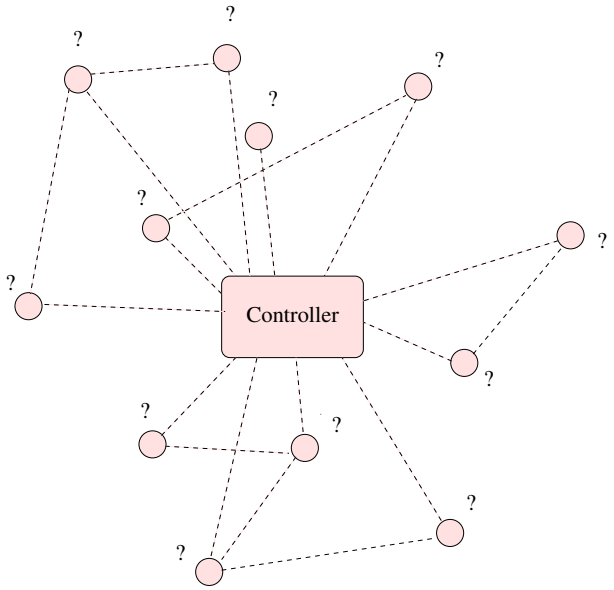

Figure 4.6: Model 4. As in Model 3, except the hosts can choose to disregard or replace aspects of policy at their option. Question marks indicate a freedom of hosts to choose.

enforcement challenges faced by Model 4 are the same (in terms of scaling properties) as for Model 3: i.e. I_{fail} is the same as that in Model 3. Hence this model can in principle work to arbitrarily large N. Model 4 is the model used by cfengine[Bur95, Bur04c]. The largest current clusters sharing a common policy are known to be of order 10^4 hosts, but this could soon be of order 10^6, with the proliferation of mobile and embedded devices.

Mesh, with partial autonomy and hierarchical coalition

An embellishment of Model 4 is to allow delegation of responsibility and overriding of central politics in a hierarchy of management. Local groups of hosts may now form policy coalitions, that serve to their advantage. A policy coalition is a region of the network that agrees on policy. Such groups of hosts might belong to one department of an organization, or to a project team, of even to a group of friends in a mobile network. Once groups form, it is natural to allow sub-groups and thence a generalized hierarchy of policy refinement through specialized social groups. If policies are public knowledge then the scaling argument of Model 3 still applies since any host could cache any policy; but now a complete policy must be assembled potentially from several sources. One can thus imagine using this model to distribute policy so as to avoid contention in bottlenecks, since load is automatically spread over multiple servers. In effect, by delegating local policy (and keeping a minimal central policy) the central source is protected from maximal loading.

Specifically, if there are S sub-controllers (and a single-layer hierarchy), then the effective update capacity is multiplied by S. Hence the threshold N_{thresh} is multiplied (with respect to that for Model 3) by the same factor. This model can be implemented using cfengine, with some additional scripting.

Mesh, with partial autonomy and inter-peer policy exchange

The final step in increasing autonomy is the free exchange of information between arbitrary hosts. Hosts can now offer one another information, policy or source materials in accordance with an appropriate trust model. In doing so, impromptu coalitions and collaborations wax and wane, driven by both human interests and possibly machine learning. Such coalitions can be stable if they have sufficient numbers to withstand outside influence or competing

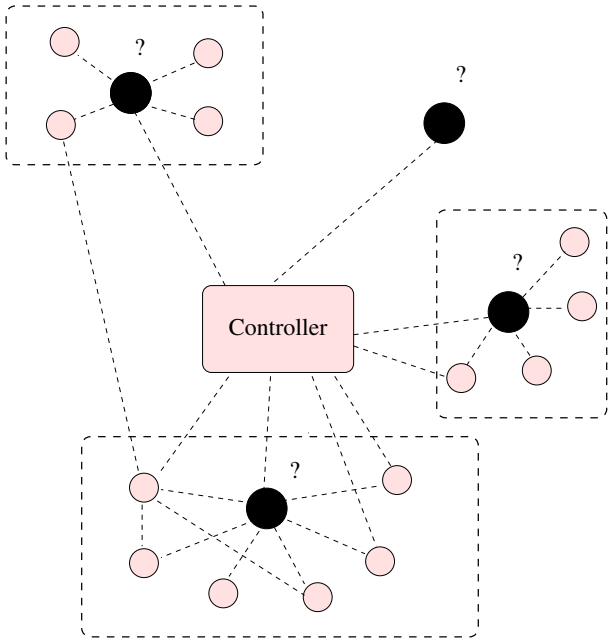

Figure 4.7: Model 5. Communication over a mesh topology, with policy choice made hierarchically. Sub-controllers (dark nodes) edit policy as received from the central controller, and pass the result to members of the local group (as indicated by dashed boxes). Question marks indicate the freedom of the controllers to edit policy from above.

influences. All we require is the existence of a stable equilibrium in graphical terms (for an explanation see ref. [Bur04a]).

A peer-to-peer policy mechanism of this type invites trepidation from those who have invested traditionally in control mechanisms, but this need not be a concern. The peer equilibration model is really no more than a distributed genetic algorithm. With appropriate constraints it could always be made to lead to sensible convergent behaviour. Such constraints can be built into policy or can be implemented as standards that constrain allowable policies. The difficulty in this 'free market economy' view of policy exchange is that the very freedoms that peer to peer preserves make it impractical to be able to guarantee a standard for policy transmission. Each device has a potentially different 'master' determining its policy. Without special care, communication between devices can lead to spreading of policies throughout a network and therefore a loss of the illusion of 'control'. There is a strong possibility that the policy network will then self-organize rather than behaving in a manner that can be considered 'managed'. In such a situation, it becomes critical to know what the stable policies for local behaviour are: there would be no sense in promising a policy that would immediately fall apart.

Example 22 *A possible way to reduce this uncertainty is to employ a communications protocol based on only voluntary acceptance. A test model of this kind of inter-peer exchange has been implemented in cfengine[Bur95] using a transport protocol called Voluntary RPC[BB05]. In this scheme hosts can safely exchange services and information without committing themselves to any reciprocal behaviour. All collaboration is entirely on a voluntary basis.*

Example 23 *One example of a collaborative peer network that has led to positive results is the Open Source Community. The lesson of Open Source Software is that it leads to a rapid evolution. A similar rapid evolution of policy could also be the result from such exchanges. Probably policies would need to be weighted according to an appropriate fitness landscape. They could include things like shared security fixes, best practices, code revisions, new software, and so on. Until this exchange nears a suitable stationary point, policy updates could be much more rapid than for the previous models. This could potentially dominate promise-keeping behaviour.*

The peer model has no centre. Hence it is, by design, scale-free: all significant interactions are local. Therefore, in principle, if the model can be made to work at small system size, then it will also work at any larger size. In practice, this model is subject to potentially large transients, even when it is on its way to stable, convergent

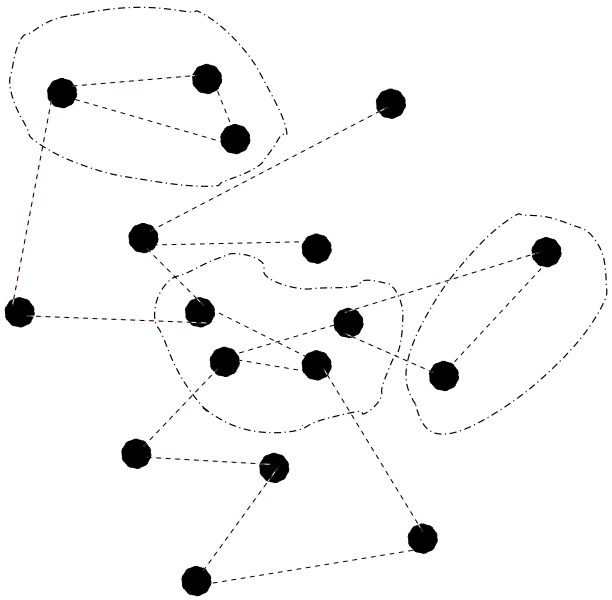

Figure 4.8: Model 6. Free exchange of policies in a peer-to-peer fashion; all nodes have choice (dark). Nodes can form spontaneous, transient coalitions, as indicated by the dashed cells. All nodes can choose; question marks are suppressed.

behaviour. These transients would likely grow with the size of the network. Here we have confined ourselves to long-time behaviour for large N—hence we assume that the system can get beyond such transients, and so find the stable regime.

When policy is determined by ad hoc collaboration many questions arise. Do such ad hoc policies form stable regions, functionally equivalent to centrally enforced star networks, or will the concensus in a system fall apart? Are such regions stable to 'infection' by mutant polices? How stable do they need to be? In ref. [Bur03] it is argued that policies should be based always on fixed points or equilibria of possible host strategies, since these are the regions that provide the necessary stability for hosts to perform a reliable function. Studies of destabilization of policies by successful mutant policies are a matter for game theoretical studies[Axe97]. We aim to return to these in future work in more detail. For now, we only point out that there is a graph theoretical phenomenon to be analyzed when policy is allowed to communicate itself from host to host. In a real sense, the dynamics of policy become directly analogolous to the dynamics of bioliogical diversity: DNA spreading and viral mutuation. Policies become indestinguishable from viruses and the likelihood of their dominance in a network ecology depends on the large scale structure of the network graph[BCE04].

4.3.2 Event-based workloads, arrival processes, and queues

Faults and repairs do not really happen according to a constant current, though this is a convenient simplification that helps us to gauge mean behaviour over long times. Rather, faulrs occur sporadically as events. Events arrive a 'random' times. A constant current is an approximation that is equivalent to a Poisson arrival process.

An arrival process is a series of random arrivals, distributed in time, with average arrival rate λ is processed serially by a server at processing rate μ. Think of rain falling (a random process of arrivals), and being led away down a drain. All we are interested in, over a steady state in time, is how much rain falls and whether we can drain it away quickly enough by introducing a sufficient number of drains (processors or servers). Thus scalability is usually discussed as throughput as a function of load. Sometimes the input is a function of a number of clients, and sometimes the output is a function of the number of servers (see fig. 4.12). There are many ways to talk about scaling behaviour.

Queueing theory deals with an abstracted model of queueing. A queue is a serial process of arrivals that is processed by a server. There are many models of queues, with arbitrary complexity, but there are some simple conclusions that summarize the pertinent results for most purposes. The simplest queueing results are for Poisson distributed arrivals and process dispatches, but they can be generalized to other distributions. For all the mathe-

matics, there are two results that are important here:

1. The average response time between sending an arrival into a single queue, and its successful processing by a single server is:

$$R = \frac{1}{\mu - \lambda}. \tag{4.14}$$

2. The traffic intensity λ/ρ is the dimensionless scaling parameter that controls queue behaviour.

3. For Poisson arrivals, this is called the M/M/1 queue, designating arrival/dispatch/servers, where 'M' stands for Markoff process.

4. As the arrival rate approaches the servicing rate, the response time grows to infinity very suddenly (power law scaling), and the system becomes unstable (see figure 4.9).

5. As the queue reaches instabilty, the response time ceases to grow linearly and the server CPU rate becomes saturated (see figure 4.10) as processes contend over the shared CPU resource.

6. Thus the queue demonstates the contention for throughput, when filling an interval of time with random arrival events.

A single queue can only promise its best service rate, so it can only be 'scaled up' to higher workloads by improving the service rate μ. However, if events can be handled in parallel, then we can also add more agents to promise the service, at the possible expense of making them work together. In the simplest form, queueing theory assumes

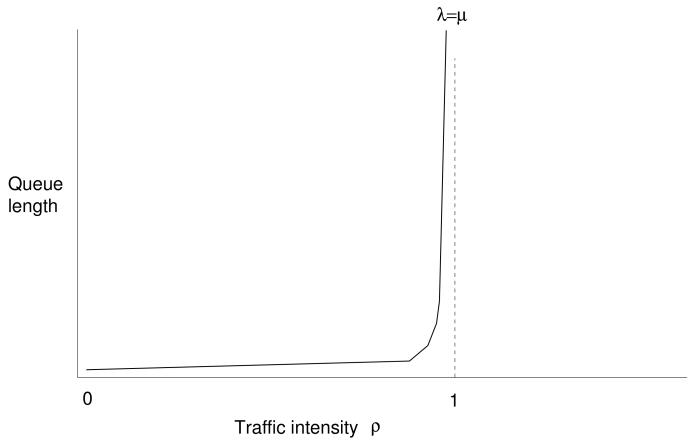

Figure 4.9: A queue behaves well when the arrival rate is far below the service rate, then it undergoes a rapid collapse into thrashing contention.

that all jobs are of equal magnitude, and the system is completely homogeneous. It is a simple order of magnitude estimate. However, what it does is to expose the existence of an instability, or a critical dimensionless ratio of arrivals to dispatches that causes the linear processing of the system in time to blow up, or fail to keep its promise of a timely completion.

4.3.3 Scaling throughput with multiple servers

We can add more servers in one of two different ways (see figure 4.11). Either we multiply the whole system by a factor of n, i.e. have n queues with n servers, and arrivals enter one of these at random. This is randomly chosen parallelism, called (M/M/1)^n. The other alternative is to have a single queue being dispatched by n servers working in parallel, so all arrivals enter the same line and are processed by the first available server. This is called the M/M/n queue.

It is a provable assertion that the $M/M/n$ is never worse and often faster than the $(M/M/1)^n$, because severs can be kept busy all the time by marshalling the flow. However, we should not overstate the importance of this.

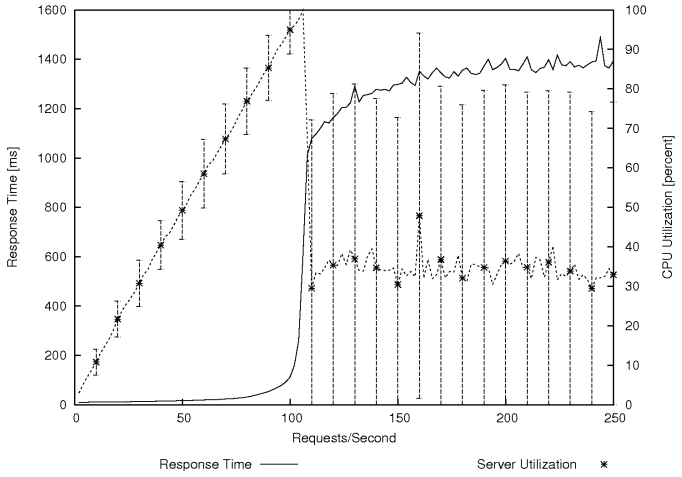

Figure 4.10: A queue behaves well when the arrival rate is far below the service rate, then it undergoes a rapid collapse into thrashing contention.

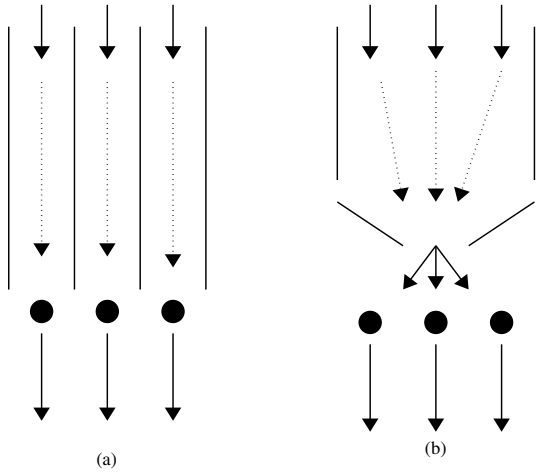

Figure 4.11: Multiple servers can be introduced in two ways: (a) by replicating the entire queue n times, and (b) by making a singlr queue services by n servers. Of these (b) is always has the shortest average response time, but it introduces a bottleneck and a longer single queue.

It was shown that a simple voluntary load balancing technique, either round robin push or voluntary server pull could perform as well as, or better than $M/M/n$[BU06, BB08] in practice, since the benefits occur mostly close to saturation. The latter is the benefit of a pull based architecture, where the servers pull from the queue when they can. Pushing data into a random queue risks entering a busy queue, while other queues are empty and idle.

The serial loadbalancer or dispatcher architecture is a common design in all kinds of services. It has a basic flaw which is the serial bottleneck introduced by the load balancer. It could be avoided by direct routing of traffic to servers.[3]

4.3.4 Amdahl's law for parallel processing

Another order of magnitude estimator is Amdahl's law[Amd67]

Amdahl's law, named after computer designer Gene Amdahl, was one of the first attempts to characterize scalability of tasks in High Performance Computing, in relation to the number of processors[Amd67]. It calculates the expected 'speed-up', or fractional increase in performance, as a result of parallellizing part of the processing

[3]DNS round robin load balancing works in this way, as a directory service. The simple average round robin balancing algorithm works almost as well as more determinisitic feedback algorithms at very low cost[BU06].

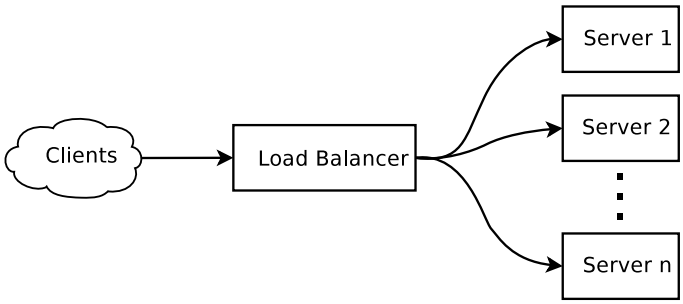

Figure 4.12: The serial loadbalancer or dispatcher architecture is a common design in all kinds of services. It has a basic flaw which is the serial bottleneck introduced by the load balancer. It could be avoided by direct routing of traffic to servers.

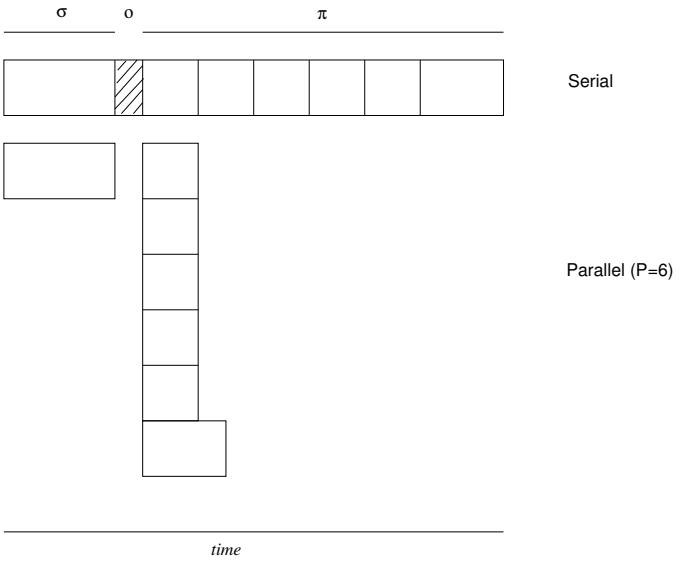

Figure 4.13: Representation of Amdahl's law. A typical task has only a fraction that is parallelizable. The serial part cannot be sped up using parallel processing or load balancing. Note that one is not always able to balance tasks into chunks of equal size, moreover there is an overhead (shaded) involved in the parallelization.

(load balancing) between n servers or processors as:

$$S(N) = \frac{t_{\text{serial}}}{t_{\text{parallel}}} = \frac{T(1)}{T(N)} \tag{4.15}$$

Suppose a task of total size $\sigma + \pi$, which is a sum of a serial part σ and a parallelizable part π, is to be shared amongst N servers or processors and suppose that there is an overhead o associated with the parallelization (see fig. 4.13). Amdahl's law says that:

$$S(N) = \frac{\sigma + \pi}{\sigma + \frac{\pi}{N} + o}. \tag{4.16}$$

In general, one does not know much about the overhead o except to say that it is positive ($o \geq 0$), thus we write

$$S(N) \leq \frac{\sigma + \pi}{\sigma + \frac{\pi}{N}}. \tag{4.17}$$

This is usually rewritten by introducing units of the *serial fraction*, $f \equiv \sigma/(\sigma + \pi)$, so that we may write:

$$S(N) \leq \frac{1}{f + \frac{(1-f)}{N}}. \tag{4.18}$$

This fraction is never greater than $1/f$, thus even with an infinite number of processors the task cannot be made faster than the processing of the serial part. The aim of any application designer must therefore be to make the serial fraction of an application as small as possible. This is called the bottleneck of the system.

Amdahl's law is, of course, an idealization that make a number of unrealistic assumptions, most notably that the overhead of sharing a task between a number of equivalent subagents is zero. Overhead is introduced by dispatchers or intermediaries. Also, it is not clear that every task can, in fact, be divided equally between a given number of processors. This assumes some perfect knowledge of a task with very fine granularity. Thus the speedup is strictly limited by the largest chunk (see fig. 4.13) not the smallest. The value of the model is that it predicts two issues that limit the performance of a server or high performance application:

- The serial fraction of the task[4].
- The processor entropy or even-ness of the load balancing.

Amdahl's law was written with high performance computing in mind, but it applies also to load sharing for any kind of network services. It applies to superagent scaling in a promise model, up to the assumptions mentioned. If one thinks of an entire job as the sum of all requests over a period of time (just as drain-water is a sum of all the individual rain-drops) then the law can also be applied to the speed up obtained in a load-balancing architecture such as that shown in fig. 4.12. The serial part of this task is then related to the processing of headers by the dispatcher, i.e. the dispatcher is the fundamental limitation of the system.

Comment 23 (Current approximation) *Another way of looking at Amdahl's law in networks has been examined in section 4.3.1 and [BC03, BC04] to discuss centralization versus distribution of processing. In network topology, serialization corresponds to centralization of processing, i.e. the introduction of a bottleneck by design. Sometimes this is necessary to collate data in a single location, other times designers centralize workflows unnecessarily from lack of imagination. If a centralized server is a common dependency of n clients, then it is clear that the capacity of the server C has to be shared between the n clients, so the workflow per client is*

$$W \simeq \frac{C}{n}. \tag{4.19}$$

We say then that this architecture scales like $1/n$, assuming C to be constant. As N becomes large, the workflow per client goes to zero which is a poor scaling property. We would prefer that it were constant, which means that we must either scale the capacity C in step with n or look for a different architecture. There are two alternatives:

- *Server scaling by load balancing (in-site or cross-site) $C \to Cn$.*
- *Peer to peer or mesh architecture (client-based voluntary load balancing) $n \to 1$.*

4.3.5 Gunther's universal scaling law for processing

In a similar vein to Amdahl's law, there is Gunther's Universal Scalability Law[Gun93, Gun08], which was proven to be derivable from a Machine Repairman model assumption (see figure 4.14). Like Amdahl's law, it assumes that the processing is due to multi-agent processing, and is careful to define scalability rather than scaling. It generalizes the service capacity speedup function $S(n)$ there by adding the cost of coordinating the n servers in the service pool, which is of order $n(n-1)$. This corresponds to the overhead term in 4.16.

$$S(n) = \frac{n}{1 + \alpha(n-1) + \beta n(n-1)} \tag{4.20}$$

In principle there could be higher order terms too, but these become decreasingly relevant to performance (they may affect semantics). For $\beta = 0$, this reduces to Amdahl's law. The denominator terms describe, a constant term that scales with the parallel concurrency of the numerator; an order n term, with coefficient α, which describes contention between the servers; and an order N^2 term, with coefficient β, which describes consistency (coherency) and network coordination between the parallel servers[5], i.e. the cost of equilibration of cached data (see figure 4.15). We see sub-linear scaling due to the cost of coordination.

Interestingly, superlinear scaling has been identified with negative values of α the model. These negative values are effective values in the scaling relation that are fed by recovery of an inefficiency elsewhere[6]. The finiteness of

[4] It is often quoted that a single woman can have a baby in nine months, but nine women cannot make this happen in a month.

[5] Paxos, Raft, and other consistency algorithms involve processes of this type.

[6] The appearance negative contention can manifest when contentions between agents for common resources are mitigated. This effect has also been observed in the scaling of cities[Bet13].

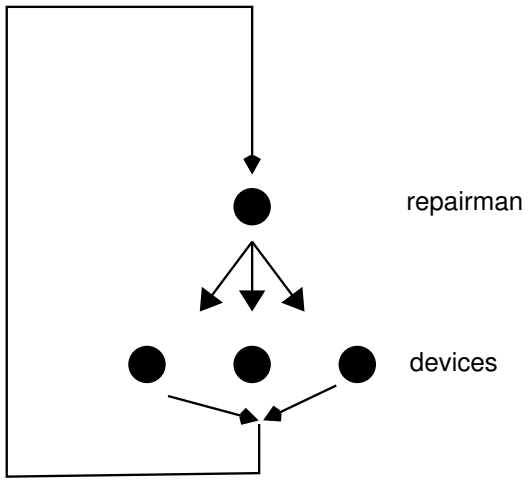

Figure 4.14: The machine repairman model is a queue where N parallel producers of traffic are handled by a single queue for repair.

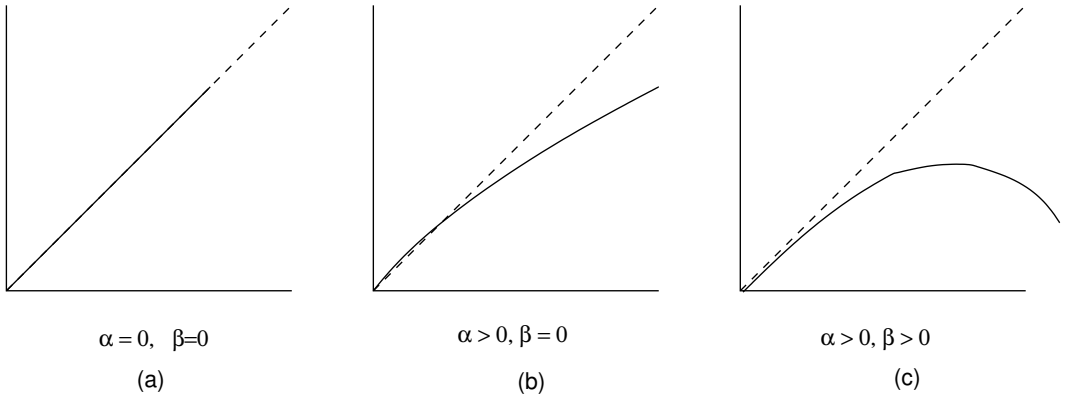

Figure 4.15: The Gunther universal scaling law for its control parameters. (a) linear scaling, (b) cost of sharing resources and diminishing returns from contention, (c) Negative returns from incoherency and the cost of equilibration.

system resources seem to indicate, however, that initial superlinearity may have to be paid back with a quadratic term later, which indicates that the superlinear graph must cross back into a sublinear and eventually degrading performance cost for large superagent clusters. Thus allowing large superagent scaling in a system could itself be a design flaw[7].

4.3.6 Effective power law scaling from Amdahl's and Gunther's law

The Amdahl and Gunther scaling relations are type 1 workload scalings, not universal scaling relations. Let's consider how we could derive a type 2 scaling relation from these.

Considering the relative speed up is a point of view that is mainly of use in parallel computation. If we want to know how the time fraction (speedup) scales for as a function of the number of processors N, then we can compare

[7] It is interesting how there is a limit to social networks due to limited brain size cost, as well known from the Dunbar hierarchy[Dun96, ZSHD04].

N with γN. Then, we can write, for $\gamma > 1$

$$\frac{T(\gamma N)}{T(N)} = \frac{\sigma + \frac{\pi}{\gamma N}}{\sigma + \frac{\pi}{N}} \tag{4.21}$$

$$= 1 + \frac{\left(\frac{1}{\gamma} - 1\right)\frac{\pi}{\sigma n}}{1 + \frac{\pi}{\sigma n}}. \tag{4.22}$$

Let $\delta = \frac{D}{D+H}$, where $D = 1$ and $H = \frac{\pi}{\sigma n}$, then, approximating as a binomial expansion,

$$\frac{T(\gamma N)}{T(N)} = 1 - \left(\frac{\gamma - 1}{\gamma}\right)\delta. \tag{4.23}$$

$$\simeq \left(1 - \left(\frac{\gamma - 1}{\gamma}\right)\right)^{\delta} \ldots \tag{4.24}$$

$$\simeq \gamma^{-\delta}. \tag{4.25}$$

Thus we have an approximate power law fit for large N compared to π/σ, and we can write

$$T(N) \simeq T_0 N^{-\delta}. \tag{4.26}$$

i.e. there is a marginal relative economy of scale for small N, which decays to an essentially scale invariant constant result. If we allow, like Gunther, for the presence of equilibration, or mesh coherence effects, then we could use the form

$$T(N) = \sigma + \frac{\pi}{\gamma N} + \kappa N, \tag{4.27}$$

where κ represents linear time take to poll each of the worker agents. This is the case where replication and consistency are required. With this extra term, we have

$$\frac{T(\gamma N)}{T(N)} = \frac{\sigma + \frac{\pi}{\gamma N} + \kappa\gamma N}{\sigma + \frac{\pi}{N} + \kappa N} \tag{4.28}$$

$$= 1 + \frac{\left(\frac{1}{\gamma} - 1\right)\frac{\pi}{\sigma} + (\gamma - 1)\frac{\kappa}{\sigma}N^2}{1 + \frac{\pi}{\sigma} + \frac{\kappa}{\sigma}N^2}. \tag{4.29}$$

Now the behaviour doesn't separate cleanly, and there are two regimes of approximate power law scaling, with something more messy in between. Using the same procedure as before, we get an anomalous term for $\kappa \neq 0$:

$$\frac{T(\gamma N)}{T(N)} \simeq \left(1 - \left(\frac{\gamma - 1}{\gamma}\right)\right)^{\delta} + \frac{\kappa(\gamma - 1)N^2}{\pi + \sigma N + \kappa N^2} \tag{4.30}$$

$$\simeq \gamma^{-\delta} \quad (\kappa \ll \sigma, N\text{small}) \tag{4.31}$$

$$\simeq \gamma \quad (\kappa > 0, N\text{large}) \tag{4.32}$$

$$\tag{4.33}$$

So for large N, with $\kappa > 0$, we have simply

$$T = T_0 N, \tag{4.34}$$

i.e. the scaling cost becomes linearly worse with increasing size.

When we compare these results to a spacetime scaling, it will become apparent that this takes the approximate form of a scaling law in a one-dimensional spacetime $D = 1$, with Hausdorff dimension $H = \pi/\sigma n < 1$. This indicates that a serial workflow, with some parallelism, is essentially a one dimensional problem, with some fractal complexity in its trajectory due to parallelism[8]. Interestingly, as the parallelism increases, the duration of the fractal dimensionality shrinks to nothing. Thus the large N limit for serial processing tends to squeeze the degrees of freedom in the system. We can compare this result to the case of general spacetime involvement in section 4.4.3.

[8]Alternatively, if we think about the problem graph theoretically, we can also say that it behaves like a $D = N$ dimensional space, and a trajectory with Hausdorff dimension $H = \pi/sigma$. In a graph, the node degree $k = N$ is the effective dimension of spacetime at the point[Bur14].

4.4 Type 2 (universal) scaling and dynamical similarity

Scale invariance, allows us to characterize dynamical properties of a system as an approximately universal and continuous function of some scale parameter. Type 2 scaling is an invariant characteristic, i.e. it is not tied to a specific scale (Amdahl's and Gunther's laws compare values at N threads to a single thread rather than to an arbitrary increase in size. We saw that these laws are not scale invariant in section 4.3.6. In type 2 scaling, we return to the idea of a continuum flow approximation, and consider its generalization from one dimensional flows to the include the involvement of more spacetime dimensions.

The three cornerstones of output scaling are still the same as for type 1, i.e.:

1. Serialization of workflow.
2. Parallelism (multiple channels to increase throughput).
3. Multiplexing processes to interleave sparse arrival processes.

What may be surprising to information technologists, who work exclusively with type 1 scaling, is that the behaviour of a system can involve more than one spatial dimension, and result behaviour that is less like a queue and more like a funnel. The involvement of spacetime geometry in processes is not something that is currenty modelled in information technology. In a building, or a city, where people can move in two or three dimensions, the dimensionality is critical to avoiding contention and enabling throughput. Consider the evacuation of a building through a single emergency exit on the ground floor, and compare this to a queue to enter the building.

4.4.1 Space and time

All non-trivial systems develop in time, but some are isolated from their surrounding space. When a system is embedded in space and time, the geometry of the surrounding space constrains the flows that feed input and deliver output.

We should not be too blinkered by queues: the narrative we have created around workflow is linear, mainly because human thinking in linear, but there are plenty of examples of systems who input or output are multidimensional:

- Water flowing into a drain from a 3 dimensional reservoir.
- Animals feeding around the carcass of prey.
- A loudspeaker radiating sound into three dimensions.
- Traffic flowing through a street network in two dimensions.
- Aircraft lift transducing airflow in one dimension to perpendicular dimension.
- The storage of data streams in two dimensional disks or three dimensional crystals.
- The representation of one dimensional data on two dimensional screens.

Comment 24 (Spacetime dimensionality) *At high speed, an agent's world is dominated by time, and a single vector of motion. In a world dominated by time, there is only serial processing, even with rudimentary parallelism, and the world is largely one dimensional. In a world dominated by space there is multidimensionality, and volumes can quickly become relevant in a continuum limit.*

The promise of semantic space, or infrastructure, is to deliver one dimensional time saving services to its clients. That means it needs to deliver either high speed or multiplexing of resources, in the space of the clients. Our narrative about semantics, and promises, involves graphs, not spatial variables. From a graph viewpoint, multiple dimensionality promotes an increased average connectivity for the graph because more neighbours can crowd into a three dimensional region and a two dimensional region or a one dimensional region.

Mathematically, a graph's spacetime dimensionality at a point is the number of possible degrees of freedom radiating from the node, i.e. the number of independent vectors emanating from it, or the node degree k[Bur14].

However, in a system realized in the context of the physical world, graphs must be embedded in a spatial volume of at least two dimensions, or more generally D dimensions. Physical agents have physical dimensions, i.e. they take up space, and links between them may not be allowed to cross because they occupy space (see figure 4.16). This places restrictions on behaviour that do not apply in the virtual world.

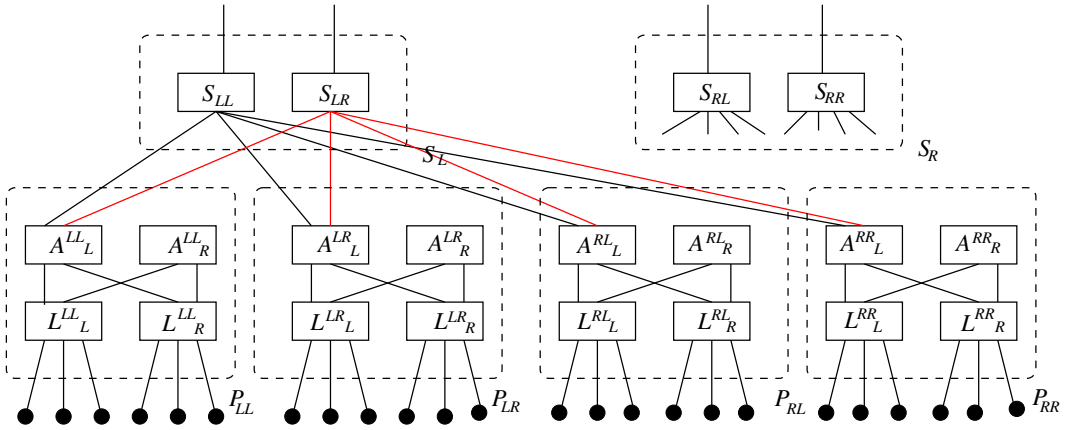

Figure 4.16: A redundant Clos fabric, takes up a three dimensional space. There is no approximation in which its behaviour can be understood in less than two dimensions.

Universal scaling relations come from ignoring the differences between labelled objects, including boundaries, i.e. by reclassifying them or counting them by lumping them into fewer categories, labelled by a scale parameter. The symmetry can be broken by the presence of a boundary.

4.4.2 Economies of scale

Outputs and characteristics of systems, which depend somehow on the scale of a system, can be related to its size, or some other dimensionless scale. The dimensionful parameters are often linked together by a functional equation, like an equation of state in thermodynamics that describes the macrostate as as relationship between the involved scales. The result for a single variable, that exhibits scale invariance is a relation of the form:

$$V(N) = V_0\ N^{\beta}. \tag{4.35}$$

This explains how the variable V may be expected to scale as a function of the number of agents, etc. This is scale invariant because

$$\frac{V(\alpha N)}{V(N)} = \frac{V_0\ (\alpha N)^{\beta}.}{V_0\ N^{\beta}} = \alpha^{\beta}. \tag{4.36}$$

The relative scaling is independent of the value of N. This was not the case in the Amdahl and Gunther laws.

The dimension of space cannot appear explicitly in this relation, due to the scale invariant form. However, it can appear in the vale for β. In fact, it can determine whether a system becomes squeezed or diluted as it scales: whether resources become burdened or plentiful. There are three cases for the scaling, as before (see figure 4.17):

- Sub-linear scaling, or economies of scale $\propto N^{\beta<1}$. Ot costs proportionally less in terms of infrastructure investment to keep a system running. For example, biological organisms use only $N^{0.75}$ more energy as their size increases, so larger organisms are more efficient. On the other hand, they also get slower because it takes longer to transport energy and signals throughout their bodies.

- Linear consumption of resources $\propto N$ (food and energy consumption). Energy is conserved so there is no way around the this.

- Superlinear amplification of output $\propto N^{\beta<1}$ (economic output of cities, etc.). When there are 'smart' components that can specialize and collaborate, to exploit the recovery of generally poor efficiencies, size can actually result in a net gain. This happens in cities, for example.

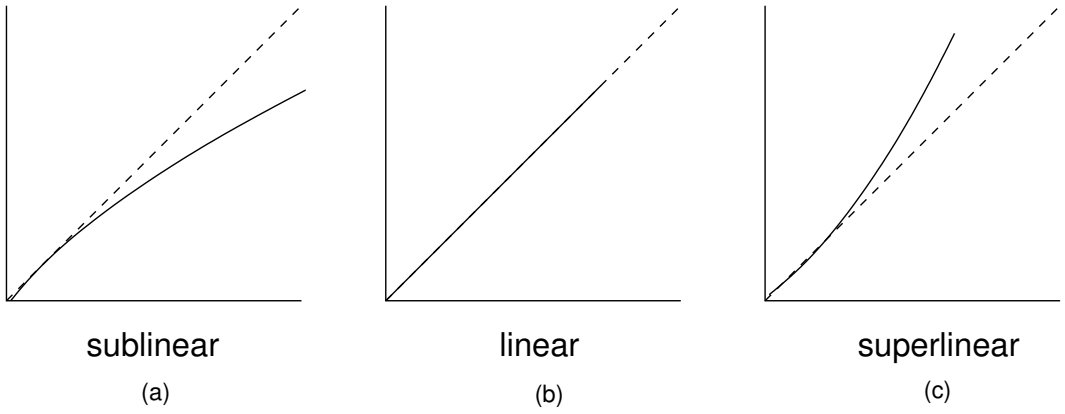

Figure 4.17: Sublinear, linear, and superlinear scaling of a value with respect to a control variable

.

Example 24 *Interesting studies have shown that cities (which are semantic/dynamic systems somewhat similar to IT systems) exhibit intriguing universal scaling, that is reminiscent of biology.*

4.4.3 Spacetime involvement in scaling

In a mean field model of systems, network links are counted using a continuum approximation in terms of volumes, and fractions of volumes. The details of agents and their promises are all averaged away. This approach has been used by [Bet13] to study the scaling in cities. It is an unfamiliar approach in computer science, but it plays an important role in deriving the scaling laws. The minimum size of the infrastructure network can be estimated by squeezing the total sparse volume into a narrow, approximately one dimensional pipeline, with a small cross section. This is only plausible of the network utilization is really sparse, since then the total interaction can be compressed into the lower dimensional network, by multiplexing. The average distance between agents inside the system (in D dimensions) is

$$d = \left(\frac{V}{N}\right)^{\frac{1}{D}} . \tag{4.37}$$

An embedded infrastructure network has structure that pervades space, because it is embedded in a real world volume, and needs to reach the homogeneously distributed agents within. This 'space filling', or fractal quality, was important in deriving the biological scaling laws[Wes99].

Example 25 *For cities, this space filling is more like an embedding than a fractal thickness to paths; however, following [Bet13], we may assume that the network fills space to some level, so that the interactions around it fill out an effective (Hausdorff) dimension $H < D$, and we may write the order of magnitude estimate for the infrastructure volume:*

$$V_I \geq g_I \left(\frac{V}{N}\right)^{\frac{H}{D}} L^{D-H} N, \tag{4.38}$$

where $g_I < 1$ is a dimensionless constant that indicates the fraction of nodes in N spanned by the particular infrastructure being considered. L is some fixed scale with the dimensions of length $[L]$, so that

$$[V] = [L]^D. \tag{4.39}$$

and $L^D \ll V_I \ll V$. In other words, the volume is the effective average linear volume swept out by a fixed cross section $L^{\frac{D_H}{D}}$, as it feeds into the N nodes connected by the infrastructure[9]*. This has the schematic form*

[9]It is well known that the scaling of ad hoc communications networks, where agents are distributed randomly is like $\sqrt{N}$. This is easily understood from the spatial geometry: mobile phones occupy some approximately two dimensional area, so the diameter is of order $N^{\frac{1}{2}}$; alternatively, they have average separation $d \simeq V/\sqrt{N}$, so the distance across the group is of the order $Nd \simeq \sqrt{N}$. The linearity of the process gets mixed up with the geometry of the embedding space.

of N queues that are serialized paths of dimension $V^{1/D}$. For $H = 0$ the nodes are unconnected, for $H = 1$ roads are serial or linear, and for $D > H > 1$, the roads or channels have an effective fractal 'thickness', from a coarse-grain perspective (see fig. 4.18). It turns out that we only need to look at $H = 1$, as it is serialization rather than physical dimension that is important.

For cities, the picture is been visualized in terms of physical channels, roads, cables, and transportation cost etc; however, any serial stream of work could constitute a virtual path for the infrastructure, coming from a number of agents within the volume V. The $V^{1/D}$ scaling represents a serialization of the work from across the homogeneous region. So, we'll show later that this also applies to services provided from spot locations, without considering the communication channel at all. Transport need not be the cost of the work, but the argument still applies to the serialization of tasks in as a queue. This means we can write the infrastructure volume as

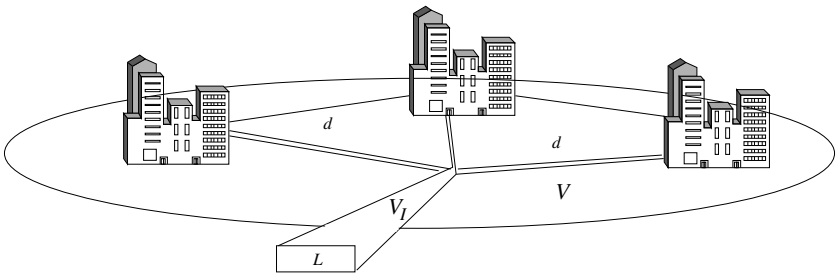

Figure 4.18: The volume of the infrastructure sparse network is negligible compared to the ball of the city.

approximately (rewriting (4.38)):

$$V_I \geq g_I V^{\frac{H}{D}} L^{D-H}\ N^{1-\frac{H}{D}}, \tag{4.40}$$

representing N_I serial queues of supporting services, being fed from a D dimensional region.

A rational queueing expression, in Gunther's Universal Scaling Law (4.15), can never explain fractional scaling exponents seen in cities, but it can demonstrate some projected superlinearity with $\alpha < 0$. To feed superlinearity, we need something more than parallel serial processes where the work is done by N point sources. Only if the work is done by N^2 interactions can a partial efficiency make the exponent greater than unity. To get this, we need to feed higher dimensional volumes into lower dimensional volumes. This is what is going on in the mean field theory of [Bet13].

$$\text{Interaction output} = \text{Maximum output}(N^2) \times \text{Fraction of infrastructure involved}(N) \tag{4.41}$$

From a graph theoretical perspective, this is a change in average connectivity of the infrastructure network (i.e. the average degree of nodes k[Bur04a]). If the fraction is a fraction of a volume rather than a line or a number, there are dimensional exponents involved, which represent the contact efficiency by close packing the city population. With more dimensions, a larger surface area can be used for interaction. If we assume that the infrastructure network is sufficiently dense that it reaches almost everyone, then this continuum approximation is reasonable.

$$\text{density of infrastructure users} = \frac{N_I}{V_I} = n_I\ N^{\delta}(D). \tag{4.42}$$

where $\delta(D) = 1/D(D+1)$, for $H = 1$. Recalling that this volume is really a continuum approximation of a network of nodes, this translates into an average node degree utilization (or locally used connectivity) within the infrastructure channel[10]

$$k(N) = \frac{N_I}{V_I} = n_I\ N^{\delta}(D). \tag{4.43}$$

[10]Note that this is not a real connectivity, which has to do with the number of nodes, but a kind of close-packing of the sparse interactions that occur between the nodes into the infrastructure stream.

Assuming the infrastructure is pervasive so $N \simeq N_I$, the equivalent serialized infrastructure volume, for a single process, is something like:

$$V_I = \left(\frac{V}{N}\right)^{\delta} \times N_I \simeq N^{1-\delta} \times \text{cross section}, \tag{4.44}$$

$$= \text{capture volume per agent} \times \text{span of agents} \times \text{cross section}, \tag{4.45}$$

Using this volume, instead of the total volume of the system (city, community, etc), recognizes two things. First that cost of the infrastructure is much less than that of the entire system; and, second, that it is the serialization of the sparse resource over a standard cross section that we want to use for comparable work output. This is like fitting the sparse output volume of the city into an idealized serial stream of fixed width to see how its length scales with the number of inhabitants[11]. The efficiency comes from being able to use more of an unexpected fixed cost, sparsely utilized resource along with other economies of scale. The net result is an amplification of the output by δ:

$$\text{Interaction related output } Y = \frac{\text{const}}{V_I} N_I^2 \simeq Y_0 N^{1+\delta(D)}. \tag{4.46}$$

The numerator is unexpectedly constant, but the infrastructure volume scales sub-linearly, the net output appears superlinear, with these assumptions. The question is how do we know if these are the same assumptions as the used for the measurements?

Modern datacentres and networks at scale have multiple redundant paths that make their interconnection networks space filling (e.g. Clos structures[Bur13], see figure 4.16). This brings higher dimensional scaling issues into the picture. The general picture is one of close packing of utilization. When dependencies scale more favorably than the contended processes that rely on them (relatively speaking), each process gets a larger share of the shared resource, and is accelerated for a while, provided the total utilization remains low.

4.4.4 The role of specialization, and modularity in efficiency

Semantic scaling and the economics of specialization

Plugins is one of the answers to software bloat, but not complexity.

Dependency is a source of fragility

Not about ease of maintenance, but about the growth of expertise and investment of time. Modularity is only useful if the different modules are supported by independent resources. Pulling a system apart to be maintained by a single person is meaningless as there is no scaling efficiency introduced.

Modularity is more expensive than monoculture unless the utilization of the modules is high. Multiplexing is what leads to efficiencies of scale.

Example 26 *Modular redesign is sometimes used as an argument for pre-planning a scaled up workforce, assuming that the workload will grow in the future. It is a gamble, because it relies on the idea that efficiencies of scale come from specialization. If the workload doesn't increase, then the cost of modularization redesign, combined with the extra overhead associated with the management of modular dependences just leads to higher costs, while the necessary extra workforce mainly sits idle.*

Example 27 *The same argument applied to the use of separate machines and containers for encapsulating IT services. the 'one service per machine' idea argued that this made management easier. It also increased the workload and the capital cost of the machinery. If the individual services were not all running with high utilization, then the independent machines were mostly idle.*

Service architecture, specialization and microservices.

4.5 Capacity of a system to invent new states that were not designed

Occasionally systems seems to invent new possibilities. We sometimes call these bugs, because the outcomes were not intended. The term emergent phenomenon is also used.

[11] A simple analogy is to think of a tube of toothpaste. The toothpaste comes out in a one dimensional stream of fixed width, but we are forcing the output of a three dimensional tube through this portal, and asking: how does the amount that comes out increase with the size of the tube if we squeeze it in the same way? By fixing the cross section, we can compare different tubes, or different cities.

4.5.1 Bugs and emergent behaviour

Definition 41 (Bug) *An outcome of a deliberately designed intentional system that was not intended.*

In the definition of a bug, we once again see that, without a promised intent, we cannot define a bug, only a surprise.

Emergent behaviours are those that seem to exhibit intentional behaviour, but for which no promise has been made.

This is the way we assocate semantics with evolutionary processes, by selection (-) promises.

We can stil try to beat these behaviours by working very fast to repair within the Nyquist frequency.

We would expect, as the scale of an intentional system grows, its ability to behave predictably diminishes somehow, as unforeseen pathological interactions become harder to predict. However, rationally, this depends on the degree of interaction not on the size per se.

What does this mean for Clos? For pods, and network design hierarchy?

4.5.2 Surprises: exploration and innovation

While systems often get more general and statistically predictable at scale, the opposite is also possible. As one mixes different parts together in a network of interactions, new phenomena occur. This turns out to be fundamental to the way ideas are generated. Although novelty can exist at all kinds of levels of sophistication, the mechanism for novelty is probably the same at all levels. It involves Darwin's key observation: mutuation, mixing and selection.

To put this in a promise theory language: independent agents go off and make changes to existing promises, then then return and recombine their efforts and select the best of the many experiments. This is branching and merging. It is sexual selection. This form of networking is the essence of a primordial soup. Perhaps it is even how new ideas form in a brain, we don't really know. It is how evolution comes up with new ideas, however: sexual mixing combined with environmental selection constraints are a form of thinking that leads to improved processes. This is innovation.

Example 28 *Open source innovation revolves largely around versioning repositories where contributors branch off copies of the current state of an idea, then modify it with atomic 'commits' to match their own experiences. Later, these changes might be merged back into the main timeline of the software, selecting those atoms from multiple experimental branches that seem to succeed best. This is sexual innovation.*

4.5.3 Mixing and separation of concerns: networks and partitions

Unintended creativity is not something we always want in systems, but it can emerge at scale. Sometimes it is useful: this is why we build teams and companies and have universities and centres of excellence. If you throw active agents together, new things will emerge[12].

Intentionality lies on top of the virtual and even physical connecivity of the effective spacetime. The connectivity represents the infrastructure networks.

Multi-tasking, or timesharing determines limits on the number of interactions an agent can have. Multi-tenancy or partitioning separates concerns into 'cells' that maintain separate accounting of resources and profits.

Interactions do not always happen directly through explicit links. They can be mediated by third parties. This is what is meant by a catalyst.

Definition 42 (Catalyst) *A 'third party' agent whose mutual interactions with other parties lead to an effective transfer of influence between them.*

This is sometimes called stigmergic interaction. For example, someone knits their friend a sweater. The sweater is seen by someone as they are waiting for a bus at the bus stop, and they ask to buy the sweater. Next week the same thing happens with a scarf. Information has been transmitted via the catalyst of the bus stop.

Formally partitioned cells are not necessarily independent, as they compete within shared boundary conditions. For example, cells might compete for the same resources, or mediators and brokers might delegate shared resources. In either case, these constraints implicitly connect otherwise intentionally separate cell networks together

[12] Like the quote from the chaotician in Jurassic Park said, 'nature will find a way'.

(weak coupling). This is why security breaches are so much easier than many expect. These 'covert channels' bridge networks. Promising to exclude is difficult. This is a 'brain model' approach.

Scaling is important because it is an enabler for opportunity: sexual mixing (male), discerning selection (female), and then post mixing isolation (gestation) in which the arduous calculations work throught their constrained narrative processes.

Random mutuation might actually be promoted by separation - isolated environments are less stable and test out mutations more quickly because in a small network, each single agent is proportionally more important (too big to fail).

4.5.4 Forces and specialized roles

A hypothesis of promise theory is that one may define a notion of force for agents, which is attractive when there is economic advantage, and repulsive for economic disadvantage[13]. The formation of superagents thus comes about, for economic reasons[BF07], by the value of collaboration. If the promises are unconditional, superagents will be localized. If they are conditional, the clusters are ordered and may thus be distended or even distributed.

- Agents, which make the same kinds of promises of same polarity, tend to repel one another.
- Agents, which make complementary (binding) promises of opposite polarity tend to attract one another.

If the hypothesis is true, one may expect behaviours such as the following:

- Dependences are held together by promise bindings.
- Competing services tend to spread out in the system region, unless they are held together by something more important.
- Distributed competitors, may cluster around a common dependence, such as shared infrastructure hubs, e.g. malls, districts.
- Chains of transport agents, bound together by conditional promises, in relay configurations.

This suggests that agents, which play the same role, will tend to move apart unless they are held together by a promise of cooperation. In promise theory, a specialized role characterizes a pattern of agents that make similar promises. By specializing specific tasks to specific agents, each agent can be more focused in learning and adapting, but acquires an additional cost of cooperation proportional to some positive power of the cluster size.

Example 29 (Data replication and cache consistency) *Consider the problem of data replication. A specialized promise, like a database, may need to work together with others to serve a wide area. Since they make the same type and polarity of promise, they tend to distribute around the region. However, they also need to cooperate to ensure that they are working together rather than against one another, with the same data. This consistency comes at the price of promising coordination. This mutual affinity for cooperation tends to bring them closer together, so they will find equilibrium orbits, where the promise forces are in balance. The resulting collaboration cluster of agents may be considered a superagent.*

Superagency collaboration is a *short range* interaction (between interior promises)[Bur15], in the sense that it is a direct agent to agent interaction.

The notions of attraction and repulsion are wired into our imaginations in terms of spatial concepts. We are used to electromagnetic and gravitational forces, like 'pressure', which seem unambiguous because we observe that a very large scales, where the affinities achieve a deterministic quality. At much smaller scales, where atoms and individual agents interact, these 'forces' are less ballistic and more probabilistic in nature. A promise model belongs at this low level scale, where forces should not be considered too literally as if they were classically deterministic.

Even without an embedding spacetime to describe vector directionality, we can speak of agent affinities, like the interactions described in molecular chemistry, where spacetime plays no real role. With a physical volume to embed a graph of promise-keeping interactions, geometry ties range to distance, but in a virtual network (which includes transport of messages by intermediate carrier), short range interactions can also be disseminated over a longer effective range, by adding cost or latency (such as in telecommunications).

[13] It does not matter here whether we consider the force to be a Newtonian deterministic force, or a probabilistic susceptibility for drifting closer, as in stochastic systems.

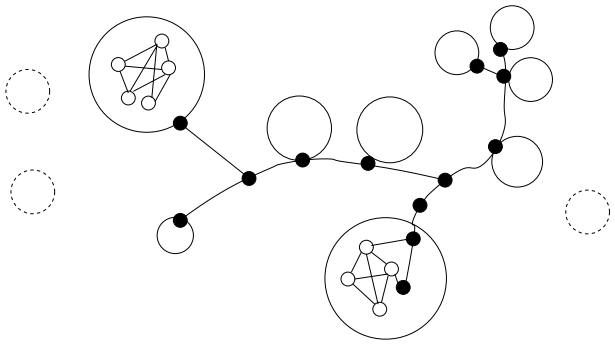

Figure 4.19: The geometry of superagents may fill space in different ways. Infrastructure that interconnects other agencies is a superagent in its own right, involving linear or approximately linear cooperation between member agents. Under preferential attachment, agents N_I tend to cluster around the infrastructure agency, leaving a few N_0 padding out the spatial volume. The circles around the subagents may be considered infrastructure binding the agents together.

4.5.5 Promise networks that percolate

Specialized promises naturally lead to small molecular clusters of agent 'atoms'. They seldom span large areas, because promises act like short range interactions, which is also why superagents can be considered quasi-atomic black box agents (see figure 4.20). General survival promises are common to most agents, and pertain to the most general kind of infrastructure in systems: power, food, air, water, etc. These are a minority of promises that are ubiquitous.

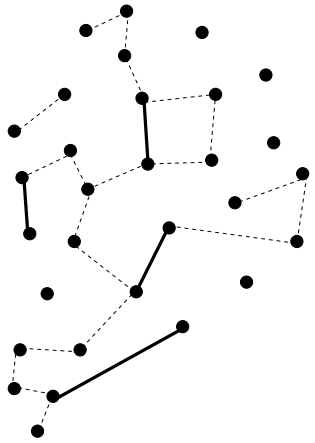

Figure 4.20: No single type of promise binding (dark lines) leads to percolation of value in the promise graph. However, with conditional dependency, and sufficient diversity and homogeneity, there can still be effectively close to N^2 links whose value converts into a common currency.

If we let N_τ be the number of agents that consume a promise of type τ, then we expect the class of τ related to survival to be of the order $N_{\text{survival}} \simeq N_I$, in the meaning of the city model. For all other types, $N_{\text{other}} \ll N_I$.

However, we'll see in section 4.5.8 that long range interactions are also needed to explain the scaling exponents for cities. The size of the effective network is not therefore given by the adjacency matrix of the underlying infrastructure network, but rather by the typed promise graph.

It is useful to recall the definition of a promise network (see [Bur15]).

Definition 43 (Promise adjacency matrix) *The directed graph adjacency matrix which records a link if there is a promise of any type τ, and body $b_{ij}(\tau)$ between the labelled agents.*

$$\Pi_{ij}^{(\tau)} = \left.\begin{array}{ll} 1 & \text{iff } A_i \xrightarrow{b_{ij}(\tau)} A_j, \\ 0 & \end{array}\right\} \quad \forall\, b_{ij}(\tau) \neq \emptyset \tag{4.47}$$

The adjacency is the effective topology of the spacetime network, as far as the agents are concerned. The link-occupancy of this matrix, for a given promise type, is a linear sum whose value is generally much lower than that of the total possible mesh of interactions. Thus, for any promise type τ,

$$\sum_{i,j=1}^{N_I} \Pi_{ij}^{(\tau)} = N_\tau(N_\tau - 1) \ll N_I^2, \tag{4.48}$$

Note that an agent can make a promise to itself too, so the upper limit could be written N_I^2.

The value-percolating connectivity or degree of a node

$$\Pi_{ij} = \sum_\tau \Pi_{ij}^{(\tau)}, \tag{4.49}$$

$$k_i \simeq \sum_j \Pi_{ij}. \tag{4.50}$$

We can also write this in terms of the direct valuation of the promises, in terms of the actual matrix of promises π_{ij}[Bur15]:

$$k_i \simeq \sum_j v_C\left(\pi_{ij}\right). \tag{4.51}$$

where v_C is the value of the promise as calibrated and assessed by a common central agency (see appendix B.2).

Agents can keep multiple promises, or multiple types, 'simultaneously' over a given timescale T, by multiplexing their time at a rate that is much faster, i.e. $\gg 1/T$ to avoid the queueing instability. On the assumption of sufficiently sparse packing:

$$\sum_\tau \sum_{i,j=1}^{N} \Pi_{ij}^{(\tau)} \leq N(N-1). \tag{4.52}$$

To understand the output of a promise network, we care more about the assessments of which promises were kept than the number of promises that were made (see appendix B.2). Each agent assesses promises individually. and they may not agree. However, to compare to city statistics, we may assume that an statistical bureau agency has been appointed by the city to calibrate these assessments $\alpha_{\text{official}}(\pi_\tau)$ according to a single scale. Promise-keeping is an average over time. Provided the sum time to keep a promise, for all τ, for each agent, is much less than each time interval of the assessments, we can reduce $\alpha(\pi)$ to a frequency 'probability'. Another way of saying this is: provided the cost of keeping the promises is less than the budget of each agent.

These estimates are maximal. The size of a functional cluster is not really related to any of these graphs, because there are semantic constraints. Functional behaviour is a strict limitation, which leads to very sparse graphs. To gauge an average measure of the total economic impact of all functional interactions, we have to assume:

- The functions are successful in driving an economy.
- The density of implicit interactions is quite high, else a given output Y will not be represented by an average mesh density.
- There are some long range interactions that make the partially connected graph totally connected on average, even if only at a low level.

In reality, the city might be partitioned into quite independent regions, leading to a modular reducible form[BBCEM10]. It guided by preconditional bindings between agents. So, if one imagines the network that delivers output Y, it may be some maximally quadratic polynomial of N_I, related to the structure function of the network.

$$\sum_\tau \sum_{i,j=1}^{N} \Pi_{ij}^{(\mathrm{Y})} = \text{Poly}(N_I) \leq N(N-1). \tag{4.53}$$

If i, j run over all the individual agents within city limits, then these matrices are sparse and fragmented for each τ. Only with sufficient diversity of promise types will the sum graph for a city output Y have sufficient connectivity

to form a process that generates output algorithmically. The same will be true in IT infrastructure for microservices and library components.

We have to add an assumption of sufficient diversity in the types of promises to our assumptions for a city, so that the average of all of them The deficiencies of certain skills were suggested by [AHF+14] as a reason for lower than expected scaling.

4.5.6 Scaling of roles and interactions

Infrastructure is an ambiguous concept. It might comprise transport technology, vegetable patches, or simply friends and acquaintances. From a promise perspective, infrastructure comprises promisers (with positive polarity) and clients or users are promisees (with negative polarity). Even this is ambiguous, since there is always a dual interpretation in which the polarities may be exchanged[BB14].

Linear consumption by agents

For promises that are requirements for survival, every agent more or less deterministically depends on some source infrastructure agent S, the number of promises is one to one:

$$\text{Number of consumed} = \sum_{i=1}^{N_I-1} \Pi_{iS}^{(\tau)} \simeq N_I - 1. \tag{4.54}$$

If the source is distributed over several agents, then this is still true:

$$\text{Number of consumed} = \sum_{s=1}^{S} \sum_{i=1}^{N_I-1} \Pi_{is}^{(\tau)} \simeq N_I - 1. \tag{4.55}$$

Thus, the number of promises needed to supply this demand is also proportional to N_I:

$$N_{\text{infra}} \equiv N_\tau \simeq N_I. \tag{4.56}$$

Economies of scale, and spacetime involvement

When agents are interacting through links, on a small scale, the chemistry of their interactions may be based on simple counting. Normally, in promise theory, we count by agent or by link. However, as the numbers of converging links become so great that counting is impractical, there is no way to liken a process to a simple Poisson arrival queue, and we resort to flow counting based on density arguments. Let's now show that this is equivalent to volume of the infrastructure V_I in [Bet13]:

Consider a number of agents N_{infra} who provide infrastructure (gas stations, supermarket, etc) for a number of clients N_{client}.

$$\pi_{\text{infra}}: \quad A_{\text{infra}} \xrightarrow{+\text{infra}\#\mathcal{V}} \{A_{\text{client}}\} \tag{4.57}$$

$$\{A_{\text{client}}\} \xrightarrow{-\text{infra}} A_{\text{infra}}. \tag{4.58}$$

Suppose that each infrastructure agent A_{infra} can promise to service $\mathcal{V}$ clients simultaneously; then, using a simple valency argument, we have a detailed balance equation for the interactions at steady state:

$$\alpha_+ N_{\text{infra}} \mathcal{V} \geq \alpha_- N_{\text{client}}. \tag{4.59}$$

Thus for simple counting of distinguishable agents, we may estimate the number of infrastructure agents needed to support a number of clients:

$$N_{\text{infra}} \geq \underbrace{\left(\frac{\alpha_-}{\alpha_+}\right)\frac{1}{\mathcal{V}}}_{\text{intrinsic}} \times N_{\text{client}}. \tag{4.60}$$

where α_-/α_+ may be interpreted as the affinity for the service, or the reciprocal compressibility. This scales linearly with the number of clients in the catchment area of the infrastructure. Moreover, there is no way, in this

detailed formulation that we can count otherwise. The only economy of scale in this arrangement is the standard linear multiplexing result for the marshalling of $\mathcal{V}$ queues into a single queue with $\mathcal{V}$ servers, noted in section 4.3.5.

However, if we now ask how to count the number of clients that can be fed into a single infrastructure agent, in a spatial volume, with dimensional multiplexing, then the best estimate is to serialize the counting, as before:

$$N_{\text{client}} \quad = \quad \left(\frac{V_{\text{catchment}}}{N_{\text{users}}} \right)^{\frac{1}{D}} C(D) \times N_{\text{users}}, \tag{4.61}$$

where we imagine a catchment volume $V_{\text{catchment}}$, containing any number of agents N_{users} who are interested in the infrastructure service, and we serialize them along a tube of constant cross section $C(D)$ (see figure 4.21). Although these numbers only apply to a small mesoscopic volume of space, in a homogenous city, this will apply

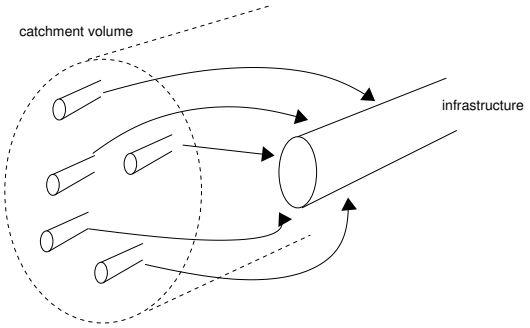

Figure 4.21: How spacetime involvement compresses serialized agent links into an effective flow of fixed cross section.

to the entire city, so we are justified in taking

$$N_{\text{use}} \simeq N_I, \tag{4.62}$$

which, combined with an equation of state for the volume, reproduces the earlier result

$$N_{\text{infra}} \simeq V^{1/D} N^{1-1/D}. \tag{4.63}$$

In this argument, it is clear that only the active agents N_I play a role in the counting, and process flow, hence this also justifies why we can assume $N_I \to N$ in [Bet13].

4.5.7 Deriving Metcalfe's law from promise networks: the importance of conditional dependency

Promise theory predicts that links represent value in the following way. Consider then the sum of all impartial promise valuations by third party C. If we assume that all agents assess the value of interactions as strictly positive, then:

$$\text{Mean value} = \sum_{\tau} \sum_{i,j=1}^{N_I} v_C(\pi_{ij}^{\tau}) \leq c \langle \alpha_i \alpha_j \rangle N_I (N_I - 1) \tag{4.64}$$

where $N_I = \max_{\tau}(\dim(\tau))$ (see appendix B.2 about promise valuations).

Note that, in spite of the quadratic appearance from the result, this is a linear sum, so it acts automatically as a linear averaging measure. Also, for any given specialization τ, the filling fraction of the promise network is likely low; thus, a key assumption is that, when properly documented, agent's specialized promises in fact depend on many others *conditionally*, forming a wide reaching network of progressively weak coupling. Conditional promises propagate the range of value interaction[Bur16, BB14]. This is the ecosystem effect.

The weakness of coupling is not a problem provided the city is reasonably homogeneous in density. If we define an effective density for the network, which describes some probable average level $\rho \in [0, 1]$ of 'intercourse' between agents (any kind of sustained relationship), then it is fair to write the value of a network of promises:

$$\text{Mean value} = \sum_{\tau} \sum_{i,j=1}^{N_I} v_C(\pi_{ij}^{\tau}) = c\, \rho\, N_I (N_I - 1). \tag{4.65}$$

provided the total density of promises forms an SCC of order N_I members. This value can be distorted from the quadratic form by significant inhomogeneity. Now, for most cities, $N_I \gg 1$ and $\rho, \alpha_i \ll 1$, so for strictly positive value interactions:

$$v \simeq c\,\rho N_I^2. \tag{4.66}$$

This is Metcalfe's law. It depends on the assumption of strictly positive value (i.e. no non-profitable interactions), and sufficient density of promises to involve everyone in the city who belongs to the infrastructure. Why is this plausible, when most specialization leads to modularity? One reason is that modularity is only a separation of scales, not an elimination of dependency: dependencies form an ecosystem. Nearest neighbours might hold the greatest semantic importance to a given function, but this reductionist viewpoint is not independently sustained without the eigenstability of the entire web[BBCEM10].

4.5.8 Conditional dependency as an explanation for multiple superlinearity classes

We can note briefly why certain occupations scale differently. In a specialization society, singular individuals or agencies rarely have all the prerequisites to complete their work. They need to collaborate and depend on others. Thus other agents are effective infrastructure relative to them. It is the accessibility of this dependency that throttles output. To drive the long range cohesion of the whole community network, specialists come to depend on specialized services (e.g. patents depend on lawyers). This leads to a number of promise configurations.

Consider four cases:

1. **Interaction scaling**: as proposed in [Bet13], for interactive value creation.

$$A_{\text{Lab}} \xrightarrow{+\text{patent}} A_{\text{observer}} \tag{4.67}$$
$$A_{\text{Lab}} \xrightarrow{\pm\text{interact}} A_{\text{services}} \tag{4.68}$$

   Patent agencies are interacting at arbitrary range with a significant fraction the total promise graph, as a part of the ecosystem.

$$Y \simeq Y_0 \left(\frac{v}{V_I}\right) N^2 \to N^{1+\delta} \simeq N^{\frac{7}{6}} = N^{1.16}. \tag{4.69}$$

   The amplified value relies on the interplay between long range mixing, and short range isolation.

2. **Scarce agent scaling**: skilled specialist experts' output is proportional to the number of skilled agents, since their queue is sparse, and not filled by a wide volume of demand. However, the same economy of scale applies to their services when they are depended on, as 'infrastructure', by others.

$$Y \simeq Y_0 \left(\frac{v}{V_I}\right) N \to N^{\delta} \simeq N^{\frac{1}{6}} = N^{0.16}. \tag{4.70}$$

3. **Interaction promises with a scarce dependency**: such as in the case of a service that depends on a source of agents to fulfill a dependency. e.g. patents can only be produced by labs that depend on the outputs of specialized R&D employees and lawyers, working in private relationships, or in secrecy. The expression in (4.81) assumes a promise configuration like that of the assisted promise law[BB14], with a main output based on a number of agents that provide input. The dependencies produce raw output, and the 'lab' agency collates and represents the collaborative mixing, e.g.

$$A_{\text{Lab}} \xrightarrow{+\text{patent}|\text{research,legal}} A_{\text{observer}} \tag{4.71}$$
$$A_{\text{Lab}} \xrightarrow{\pm\text{interact}} A_{\text{services}} \tag{4.72}$$
$$A_{\text{Lab}} \xrightarrow{-\text{research}} A_{\text{staff}} \tag{4.73}$$
$$A_{\text{Lab}} \xrightarrow{-\text{legal}} A_{\text{lawyer}} \tag{4.74}$$
$$A_{\text{staff}} \xrightarrow{+\text{research}} A_{\text{Lab}} \tag{4.75}$$
$$A_{\text{lawyer}} \xrightarrow{+\text{legal}} A_{\text{Lab}} \tag{4.76}$$

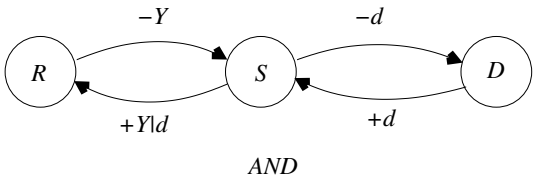

Figure 4.22: A two stage (long range) dependency has two economies of scale, when fed by a spacetime workflow. The probability of promises kept is multiplicative, like the logical 'AND' of the promises.

More generically, with two stages in the process of promise keeping, each experiencing scaling (see figure 4.22),

$$S \xrightarrow{+Y|d} R \tag{4.77}$$
$$R \xrightarrow{-Y} S \tag{4.78}$$
$$S \xrightarrow{-d} D \tag{4.79}$$
$$D \xrightarrow{+d} S \tag{4.80}$$

the total process picks up two 'economies of scale': the delivery of Y conditionally AND the delivery of the conditional dependence.

$$Y \simeq Y_0 \left(\frac{v}{V_I}\right) N^2 \times \left(\frac{v}{V_{\text{depend}}}\right) N \to N^{1+2\delta} \simeq N^{\frac{4}{3}} = N^{1.33}. \tag{4.81}$$

where $D = 2$ is used for the numerical values. These values accord better with the cited data in [BLH$^+$07], and tie in with the story about queueing.

What characterizes this interaction is the high level of specialization required to fulfill the dependencies. If the network is sparse, this is more difficult than if it is dense and diverse. This is the specialization gamble. With specialization comes individual efficiency, but also risk of instability by disconnection from key dependencies[Tai88].

They are a throttle on the process, because their absence could stop it altogether. Hence, we are justified in using the product 'AND' for combining the probably values in (4.81). This is not a hierarchical system

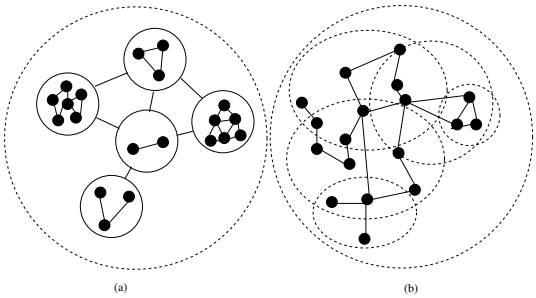

Figure 4.23: Structural recursion in an ecosystem is not like a branching process of containers (a), but rather the agents overlap with other regions of the same network to access their virtual functions. Thus their outputs are not concealed as interior substructure, but exposed as part of the flat internetwork (b). The result is that a second order recursion picks up a second economy of scale, in turn increasing the superlinearity of the derived output.

interaction, because the services are not necessarily hidden from the long range dynamics by internal components of the superagent 'lab' (see figure 4.23 (b)). But in organizational theory, one normally assumes that all organizations are hierarchically organized (see figure 4.23 (a)).

4. **Recursive promise dependency**. Let's consider what happens when the ecosystem network is based on a hierarchy of interaction ranges, i.e. promises are made recursively in fully protected shells. The agency

produces a service using full community infrastructure, but also has some specialist dependency contained entirely within (see figure 4.23 (a)).

$$A_{\text{company}} \xrightarrow{+A_{\text{specialist}} \xrightarrow{+\text{solution}} A_{\text{client}}} A_{\text{client}} \tag{4.82}$$

would be viewed as a recursive operation on the infrastructure, and the economics of scale would apply to both times the (different) infrastructures were used.

$$Y_{\text{patent}} = \frac{N_I^2}{V_{\text{patent}}} \tag{4.83}$$

$$V_{\text{patent}} = g_{\text{R\&D}} \left(\frac{V_{\text{R\&D}}}{N_{\text{R\&D}}} \right)^{\frac{1}{D}} N_{\text{R\&D}} \tag{4.84}$$

$$V_{\text{R\&D}} = \left(\frac{V}{N_I} \right)^{\frac{1}{D}} \tag{4.85}$$

Substituting for $V_{\text{R\&D}}/N_{\text{R\&D}}$ from the last expression into the former,

$$V_{\text{patent}} = g_{\text{R\&D}} \left(\frac{V}{N_I} \right)^{\frac{1}{D^2}} N_{\text{R\&D}} \tag{4.86}$$

Inserting this into the output expression

$$Y_{\text{patent}} \simeq N^{1+\frac{1}{D^2}-\frac{1}{D(D+1)}} \simeq N^{\frac{13}{12}} \simeq N^{1.08}. \tag{4.87}$$

This value is almost linear, which is what we might expect on a self-contained specialization, since the outside world would not be able to tell the difference between a single agent and a single superagent.

The value is also smaller than case (1) above, not larger, so short range hierarchical scaling cannot explain the anomalously large exponents measured in cities. On the other hand, there are some smaller exponents in this range. It is interesting to examine these measures from the perspective of the promises represented, and their range in the embedding space.

There is a simple prediction here: long range dependency seems to increase output superlinearity, i.e. dependency brings strong long range coupling and activates a larger amount of the N^2 mesh. A similar effect can be obtained in [Bet13], by slightly increasing the Hausdorff dimension of the infrastructure $H > 1$. This does would correspond to a more pervasive generic infrastructure network, which is an opaque explanation at best. It seems unclear how to justify it.

Short range dependency is basically invisible at larger scales. This observation might help to explain superlinear seen in technological contexts too, through coordination[GPT15], but we have to be careful not to mix together effects that come about due to higher dimensionality, with other mechanisms for increasing the utilization of dependent resources.

4.5.9 Service discovery of dependencies in a network

The ability to discover and match with agents, to form valuable bindings, depends on either physical or virtual mobility of the agents. Kinetically, agents may follow a random walk, as in ballistic discovery. A second possibility is that cheap intermediaries perform the discovery of specialized roles[14]. We can say that two agents are either

- Physically close.
- Virtually close.

Promise theory also predicts that they may be dynamically close or semantically close, such as when related meanings are grouped. The former depends on the length scales of the system (e.g. city) and its structure. The latter can be assumed approximately independent of these scales, because the carriers are very light, cheap, or fast

[14] Electrons play this role in molecular chemistry, or telecommunications in the human realm.

(or, in the case of semantic distance, purely cognitive). If the cost of discovery can be neglected, the cost equation is different: collaboration can be cheaper, and the value of being in close proximity for a particular specialization is reduced[15].

Directories, maps, and indices[Bur15] are ways for agents to virtualize discovery of dependencies, and locate one another at low cost. Telephone directories map coordinate addresses to names. Yellow pages map coordinates to specializations. Similar specializations are grouped. Shopping malls and industrial estates are physical directories, where clients can expect to find services in a small volume. Directories may be discovered themselves, or formed by voluntary registration. The value of new bindings overcomes the tendency for similar specializations to repel one another: similar agents may be attracted implicitly (covalently) by the intermediate attraction to clients.

The scaling estimates of the city are based on infrastructure where physical motion of the population is based on the cost of traversing some fraction of the length of the city. We can repeat the output calculation to neglect this cost, as is the case in services that do not require physical transport.

- **Physical interaction** (transport/mobility): people move around using transport infrastructure to experience their environment. There is a promise for people to observe their surroundings, for something related to subject τ, and this promise is kept fractionally $\alpha_\tau \in [0, 1]$ during their walk.

  Let the linear range of the agent A_i be some dimensionless fraction per unit time rT_{explore} of the size of the city $V^{1/D}$, where r is the speed in units of city size[16]. If the density of impulses per unit length of city region $\mathcal{I}$ is assumed constant relative to the transport rate (because this is the basis of commerce, i.e. what the city is trying to optimize for people's finite time), then the number of impulses I_τ of type τ, experienced on such a walk, may be written:

  $$I_\tau \propto r_i \, T_{\text{explore}} \, V^{1/D} \, \mathcal{I} \, \alpha_\tau \tag{4.88}$$

  where α_τ is the probability that the person or agent will be receptive to impulses in its environment that are relevant to promises of type τ.

  Although there is room for inhomogeneous variations in the city regions, in the transport rate r, and the density of offerings $\mathcal{I}$, this will not change the average scaling argument much, as long as N is large. I make the assumption here that the density of experiences $\mathcal{I}$ is constant, even though the density of people is related to the city size. This is because the size of a city is constrained by the time rather than the distance (and we are suppressing explicit time).

  The range will be some fraction of the size of the city, available by transport infrastructure $V^{1/D}$. The cost of physically fishing for ideas thus takes the form

  $$C \simeq c_Y N_I V^{\frac{1}{D}}, \tag{4.89}$$

  in agreement with the work model of [Bet13]. This applies for physical city interactions, and leads to the same output scaling expression in [Bet13].

  $$\begin{aligned} Y_Y^+ &\simeq N^{\frac{2D+1}{D(1+D)}} N_I^{\frac{D^2-D}{D(D+1)}}, & (4.90)\\ &\simeq N_I^{\frac{7}{6}} \left(1 + \frac{N_0}{N_I}\right)^{\frac{5}{6}}. & (4.91) \end{aligned}$$

- **Virtual interaction** (teleport/messaging): people are immobile and send messages to one another, watch entertainment, browse, read, talk, etc. These activities occupy an increasing amount of the time spent by people, not least because it can easily be interleaved with work time. The rate is no longer related to the size of the city, nor is there any obvious boundary to what can be discovered online (since the range of the Internet is even more diverse than a city)[17]. In this case, the impulses are more likely to be related to

[15] A dependency does not just have to be discovered, but also maintained in a persistent relationship, which accumulates cost over time.

[16] If the person's path is detailed, one could include the Hausdorff dimension of the path and use $v_i^{H/D}$ as the range, as Bettencourt suggests. I'll ignore this for now, as humans do not tend to move in fractal paths, as his data suggest.

[17] Because telecommunications networks are global, it does not make sense to relate their cost to the size of the city (though this depends on exactly how we model the costs), so the cost depends more on its usage than on its extent. We simply assume that it exists and has sufficient capacity for the N_I connected residents.

availability of the fountain itself (e.g. 'bandwidth' B) multiplied the time spent.

$$I_\tau \propto B\, T_{\text{explore}}\, \mathcal{I}\, \alpha_\tau \tag{4.92}$$

Discovery of information is the main issue. Before search engines, there were only directories such as white pages (by person) and yellow pages (by promise type). However delivery of what is discovered might still involve spatial constraints, e.g. locating a new car online does not allow it to be teleported to the buyer's location. However, 3d printing technology might change this, for a class of problems, soon.

Here it is not the locations that matter, but the rate at which impulses are absorbed. Once again, this is constant. When friends, books, or movies are communicating ideas to us, this happens at a rate that depends only on how quickly we can get hold of a stream. How users discover locations online, or by telephone is a separate question. Directories[Bur15], advertisements, and chance all play a role here. The cost of fishing for ideas is thus now independent of the city size. For a community of multiple super-agents, the analogous expression is:

$$C \simeq cN_I BT_{\text{explore}}. \tag{4.93}$$

If we imagine a community with no other infrastructure except its telecommunications network, and substitute (4.93) into the detailed balance equation:

$$g_Y \left(\frac{v_Y}{V}\right) N_I^2 \geq cN_I BT_{\text{explore}}. \tag{4.94}$$

Following through the calculation for the yield estimate identically, we find the scaling is no longer super-linear ($D = 2, H = 1$)

$$Y \simeq N^{\frac{H}{D}} N_I^{\left(1-\frac{H}{D}\right)} \simeq N_I^{\frac{1}{2}} \left(1 + \frac{N_0}{N_I}\right)^{-\frac{1}{2}}. \tag{4.95}$$

This simple result reflects the intuition that, if we neglect the 'universal cost' of telecommunications from the community accounting, then the value generated as a result of collaborative processes is proportional only to the fraction of participants who span the diameter of the city or community. This reproduces the well-known result for mobile ad hoc networking (MANET)[BC04, BC03].

The rate of output based on trawling of ideas and gestation in closed workgroups will be

$$I_\tau^{\text{work}} = N_D \left(c_{\text{phys}} I_\tau^{\text{phys}} + c_{\text{virt}} I_\tau^{\text{virt}}\right) \tag{4.96}$$

and for the entire city of N_W workplaces:

$$I^{\text{city}} = \sum_\tau I_\tau^{\text{work}} \simeq N_W \bar{I}^{\text{work}}. \tag{4.97}$$

4.6 Architecture and topology of systems

4.6.1 The potential benefits and fragility of centralization

A centralized controller allows

- Fast trusted communication inside the controller.
- Immediate correlation of information already in the controller.

The potential for centralized control is thus to make a 'brain', i.e. a place in which information can be compared and correlated, mixed and matched, very quickly.

Making the outside system depend on the results of the controller's decision:

- Single point of failure dependency
- Propagation time to reach point of action could mean latency renders response too late

- Signal has to pass through intermediate agents, which means it cannot be trusted
- Point of action can become partitioned (cut off) from the controller
- The relevance of the controller's decision diminishes with the scale of its catchment area, due to coarse graining.

The controller can only address the average properties of its catchment area, as expressed in the maintenance theorem[Bur03, Bur04a].

In order for a remote controller to be effective in the long term

Computation time + travel latency $\ll$ system response time.

4.6.2 Lessons from the study of biology and cities: viability of systems

Scaling of the generic properties of systems is an important way of examining their bulk properties. Bulk properties do not tell us everything. For example, the bulk properties of steel cannot tell us if or how a beam will fail under special circumstances, but they can tell us roughly how the average beam will behave under average circumstances. There is still value in estimates.

Cities lead to improved innovation and semantic output as a result of speed and proximity of agencies within their bounds. There are also some economies of scale by placing resources close together.

If a brain or a city can innovate, then it can offset the failure of its own coping mechanism, by resetting the timescales or moving the boundaries.

How can network properties tell us about fault or disease spreading?

How about resource transportation, leading to sustainability or viability?

4.6.3 Queueing instability

Each agent multiplexes its time between different promises it has to keep. If the time to keep the promises grows greater than its awake time, it will pass a queueing threshold, and effectively go dark. i.e. the response time of the agent will exceed the expectations of the system. Thus it will likely not keep some or any of its promises.

This is a collective failure mode that can happen as a network reaches capacity.

4.6.4 Systems with fixed and mobile agents

Fixed semantic relationships and their relative dynamics (organ plumbing), versus adaptive mobile semantics (immune cells) with variable dynamics brought to different locations in the fixed infrastructure.

Like the immune system

Mobile phones, people in cities

4.6.5 Network centrality and eigenstates

Eigensystems can tell us whether these are intrinsically stable or fragile

Can we assign more responsibility for keeping promises to certain places inside a network?

Proportional to the number of options or alternative paths an agent has available to keep its promise.

4.6.6 Percolation

We expect the possibility of cascade failures and instabilities related to semantic errors

Phase transitions

.

4.6.7 Ordering brittleness

Systems that rely on strong ordering to operate are likely to be brittle, and they have poor interaction scaling properties. The processes to make changes or to collaborate on an ordered structure with many agents (persons or software) making changes do not scale well, because the ordering implies a set of preconditions that cannot be violated when making changes and repairs. This means there will be a bottleneck of contention for each interaction.

Moreover, if any of these are violated during the repair, then a whole cascade of ancillary promises will be broken, due to the preconditional nature of ordering promises (see section 3.4.9). Thus ordering is sensitive to brittle failure.

4.7 How does high level scaling decompose at lower levels?

If we could expose the details.

4.8 Scaling Theorem ..

Do the scales decouple for fault tolerance?

The consequence of this scaling result is that it makes sense to build scale decoupling into systems, to promote resilience. That means weak coupling and redundancy everywhere possible.

4.8.1 Scaling of agility

Agility is, to some extent, the opposite of resilience. It can also be a desirable quality.

4.8.2 Scaling of awareness

4.8.3 Scaling of feedback and repair

4.9 Definition the resilience of an entire system

The scaling problem.... can we define the resilience of super agents? See chapter 4.

What information do we lose as a consequence? Can we restore transparency with directories?

Is transparency sufficient to preict failure modes? No.... prove this.

Likelihood of failures depends on stabilty, not transparency. Our best shot is to keep systems constrained to linear or sublinear (convergent) behaviours.

4.10 Protocols as recursive promises

Protocols are promises about processes or communications. Processes or messages are assembled from building blocks that combine to build standard structures.

Definition 44 (Protocol) *A convention or pattern describing non-atomic interactions, that classifies them into families by similar structure or intent.*

The structure of a message or process may therefore be thought of as a promise, but made by what agent? There are in fact several kinds of agents of different roles at work in message transmission. There are, of course, the sender and the receiver of the message. These are two agents that promise the message as an entity. However, the message itself may also be considered a stream of agents that are embedded within the 'agency of the message' in full, each promising a certain kind of content, and each acting quite independently of the transmission of the promise. The message components thus have independent agency, and meaning, within the context of a message communicated between another kind of agent.

In promise theory, each independent part of a *message* may be represented as an agent that makes promises, embedded within the larger agency of a total message. This is sometimes call layering of protocols. The promise to send and deliver a message is a separate promise from message format and content.

Protocols can be very fragile structures, as with any rigid pattern. At the highest level, an error can be made by the sender in breaching the protocol's promised pattern (+), or by the recipient in failing to parse it correctly (-). At lower levels, there is also scope for mode detailed error:

- Content error: size type and content mismatch.
- Order of prerequisites breaches the standard.
- Misalignment of component boundaries leads to confusion. subagents within the message should be within promised bounds.

Error correction, or fault tolerance, methods may be applied to avoid transmission problems. There might be a significant number of promises at work in a protocol that need to be addressed:

- Recursive agency[18] or context referring to the protocol's alphabet and dictionary.
- Data type assumptions, subagent encodings, size and alignment.
- Compound clusters and groupings (recursion of grammar).
- Interpretation of data as a 'relative delta' or 'desired end state' representation.
- Sequence number or conditional element in an on-going conversation.
- Attribute advertisement (the + promises).
- Attribute matching (the - promises).
- Missing or corrupt constraint (promise not kept).
- Version of the pattern, if the pattern is evolving

Any one of these promises broken could lead to a fault in the system at the scale of the communicating end-points.

4.11 Materal promise networks

What we have seen so far is that promises are kept, and cooperation is won through networks of agents, whose only link may be promises.

4.11.1 Molecular (super)agents and 'Microservices'

If a set of agents (at some scale) are the atomic building blocks of systems, then small, irregular constellations of those agents, bonded by promises. may be called molecules.

4.11.2 Promise fabrics and materials

Clos networks etc

What can we say about dense collectives of services that are dependent on one another?

[18] The recursion of agency is sometimes associated with the idea of 'namespaces' in information technology.

4.11.3 Fragility along interfaces and boundaries

Interaction is what makes systems non-trival. Interactions may be cooperative, non-cooperative, or even competitive. Many faults arise along boundaries, because boundaries are, by definition, non-redundant.

Drivers and transducers that try to bring a kind of uniformity to the variety of non-standardized devices and software interfaces are amongst the most common places for bugs in software, because the lack of clear or standard promises about the format of the data means that drivers and transducers are often guessing, or 'screen scraping'. This is has dubious semantic stability.

The introduction of REST APIs is a major step forward: unlike the protocols of the previous era, there is a separation of transport layer from data channel, with a common JSON or XML markup encoding.

4.11.4 Material properties

Fragile
Brittle
Strong
etc

4.12 Continuity and renewal, testing

Continuity of processes or systems...

The so-called continuous delivery framework is about the timescale adaptation of fault correction to scaled systems[HF10].

It is the scaled generalization of the convergence method described in section 3.4.8.

4.12.1 The role of testing

- Semantic tests (acceptance)
- Dynamics tests (execution and scale tests)
- Monte Carlo search methods Chaos monkey

4.12.2 Staging, simulation, and safety nets

Recreating environments or embedding in the actual environment

4.12.3 Learning, adapting systems (anti-fragility)

Contextual adaptation

4.12.4 Structural material patterns and their resilience to faults

System design principles like serialization, parallelization, centralization, decentralization, etc, have an impact on system resilience. When we scale up systems from the bottom up, using agents in a promise-oriented model, the bulk properties of systems look very much like the properties of matter in material science.

We may begin discussing these issues from standalone atoms, building up through molecules, to chains, and multi-dimensional crystals and fabrics. All if these structures are in use in technological and human systems today.

4.12.5 Load-bearing, stability and tolerance

Cooperative structures may themselves act as dependencies or foundations for subsystems that perform the work of interest. For example, the utilities: electricity, water, etc. are needed by every home and business, but they are generally considered to lie outside of the boundary of interest. Nonetheless, in an assessment of stability, these may be key load-bearing promises.

4.12.6 Cracks

Our best chance of avoiding brittle shattering is *fault tolerance*, which can absorb cracks before they propagate. This is like putting impurities into metals to disperse stress concentrations.

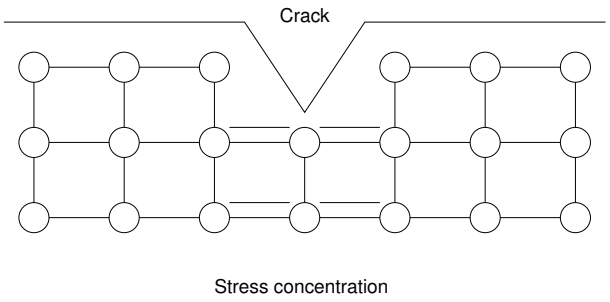

Figure 4.24: A load bearing structure with a stress concentration due to a broken or missing promise.

4.12.7 Resilient dependency at scale

Resilient dependency, from one set of agents promising a role to another leads to an interwoven. This is not the all-too-common tree structure, but rather a mesh of criss-crossed redundancy. The complexity of this grows as N^2 for a fixed level of redundancy N, hence it is expensive to implement.

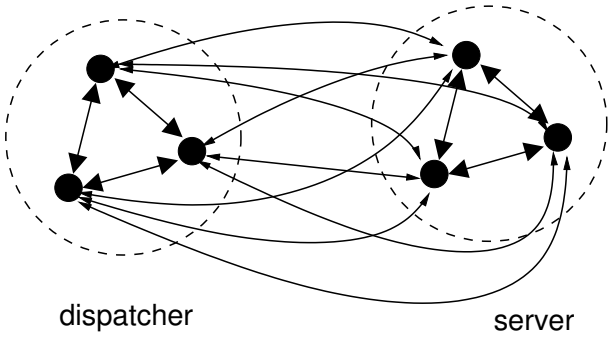

Figure 4.25: Multiple redundancy for a resilient link $n \times m$

There is another kind of failure mode, which is not associated with the loss of any component, but with the failure of the load-learing cooperative of agents.

$$\sum rates > +\text{receive rate} \tag{4.98}$$

4.12.8 Non-local auto-repair in a serial chain

If repairs need to be carried out that span multiple agents, then this is a promise renegotiation problem, analogous to contract bindings. This is an $O(N^2)$ problem in worst case. Centralized management can make it $O(N)$ at the expense of a bottleneck.

(TO DO) - Matt Disney work
Chains
Matt Disney work
Timing delays, periodic frameworks....

4.13 Summary of scaling issues

1. Scaling of the abilty to discover information and promised services.

2. Exterior promise not kept.
3. Interior promise, supporting exterior promise, not kept.
4. Loss of agent transparency at scale (missing directory).
5. Disorted propagation of influence over distance (Chinese whispers).
6. Promise throughput inadequate.
7. Promise collaboration semantically deficient for the task, boundary conditions, or scale.
8. Agent weakness leading to node failure, with avalanche failure.
9. Promise takes too long at scale, because transmission scales differently to load.
10. Increased density of faults, leading to threshold 'critical mass' effect.
11. Accumulation of delays and uncertainties in multi-agent propagation.
12. Interference and diffraction of promises along different paths and cycles.
13. Unexpected (loss of) percolation of influence at higher density of agents or interactions.
14. Force density effects, attraction and repulsion of polar promises and bindings.
15. Scaling of comprehension (assessment), i.e. inability to serially process information from a bulk region.
16. Scaling of 'agility', inertia caused by the relative change in efficiency with which promise propagation occurs at scale.
17. Scaling of response to mass 'events', i.e. arrival processes (serialization).
18. Scaling of impact, i.e.

$$\text{Impact} = \frac{\text{Affected agents}}{\text{All agents}}. \tag{4.99}$$

19. Scaling of a 'fault surface' or 'attack surface' relative to system size.
20. Scaling of feedback, leading to (de)stabilization, at short range or at long range.
21. Amplification of small errors into large ones by unstable branching.

Chapter 5

Recovery, repair, and replacement

How can a system have continuity during replacements and upgrades?

When a fault arises, either spontaneously or due to an error, we generally have to resort to one of the three R's of technology: recovery, repair, and ultimately replacement. Forensic investigation (aka 'debugging') is often initiated as a meta-process to find promises not kept; it could be speeded up considerably if we always monitored the keeping of promises. It involves gathering evidence and inferring cause-relationships that could have been documented by promises.

Faults, i.e. state that conflicts with what is promised, take several forms:

- States arising from logical errors.
- States arising from random errors.
- States arising from (unexpected) emergent interactions.

We can adopt different strategies for solving a problem:

- Mitigate the damage by relieving symptoms on a regular basis.
- Fix the cause of the problem at source, aka 'root cause analysis'.

In many complex systems, it is profitable to employ both of these, though the belief in a single 'root cause' is at odds with the network nature of most real systems.

5.1 System isolation

As mentioned in the introduction (see section 1.2.1), systems do not have well defined boundaries. They are 'open' in all practical circumstances. Many of the difficulties and errors that are made about systems arise from the tacit treatment of systems as being closed, i.e. existing immutably and in isolation from external forces. However, as soon as we make use of a device or procedure, it interacts with and is altered by its environment[1]. Design flaws often revolve around the assumption that systems are only affected by intentional changes made one at a time inside a system's artificial boundary.

5.2 Promised roles versus component failure

A fault is a promise not kept. This involves and agent and a process that emanates from the agent. The fault might be a compound fault involving several promises. The consequences of the fault are not the fault itself.

We repair roles rather than components. These are agents and the promises they make. If there is redundancy, then this is part of the role super agent.

[1]The term immutable process is sometimes used in computing to refer to the idea that what happens inside a box, which only observes the outside world, does not affect the world outside the box, even though it affects the world inside the box.

5.3 Fault impact measurement

5.3.1 The in and out sets and amplification

Incoming and outgoing states.

Mimicking the notation of quantum theory S matrix, we can write it as a scattering matrix.

$$\langle out|in\rangle \tag{5.1}$$

5.3.2 Shrinking the potential fault surface (context)

Example 30 (Sudo not su -) *The Unix command* `sudo`*, which grants one time root access, is often recommended for use instead of a privileged root login. This is because it grants privilge only for the duration of the named command, preventing accidental invocation of other commands out of context.*

5.3.3 Fault impact and trajectory correction or a fault interval Δt

If a fault is allowed to exist for Δt seconds, the impact of the fault is:

$$I = A(t).....\Delta t \tag{5.2}$$

Not that this is non-linear in general, hence it is not reversible

5.4 Transactional 'atomic' change and repair

The model of secure transactions was introduced to try to establish control over change through artificial isolation. It is widely used for financial exchanges, and database writes. Transactional thinking is very common in information technology. A transaction is a promise of locally deterministic change that relies on two main conditions:

- Exclusive process isolation for an interval of time.
- Atomicity of changes, i.e. all change happens within the interval of isolation.

Transactional systems often additionally rely on a specific ordering of changes, i.e. serialization of changes, as is usual in databases. None of these conditions are generally valid in a distributed system[2], making errors due to incorrect assumptions rife.

Example 31 (Transactional security) *There are certain 'consistency protocols' (e.g. Paxos, Raft, Zookeeper, Chubby, etc) that claim to handle transactional changes consistently in distributed systems, but these apply typically only to very simple data values not to distributed state or interactions. Moreover, they have a finite reponse time to implement, and they are not fault tolerant. The assumption of process isolation is precarious and fragile.*

We tend to think that banking systems require this kind of consistency, although banks and credit card companies regularly handle fault tolerance in transactions with buffer pools, as this is an innate part of dealing with unpredictable requests. The transactional nature only helps to generate a narrative about who did what first, often as a basis for charging fees and fines.

Example 32 (Maintenance windows) *Planned downtime, maintenance windows, or repairs on the ground, enable transactional isolation by taking system parts out of service for planned maintenance. This might result in taking down the whole system, or systems might rely on reduced capacity or 'hot wiring' a bypass (e.g. in heart surgery) to work around the issue.*

[2]The use of vector clocks is common to bring a notion of consistent serialization of changes in systems, perserving order, but this is independent of the temporal consistency of global state, which is basically impossible without locking in a highly disruptive manner. Thus a promise of consistency might be given at the expense of a promise about availability. This is known as the CAP problem[Bur12].

5.4.1 Rollout and rollback

Planned change goes through a decision lifecycle. In a distributed system, the moment of committing to the change is sometimes called the rolling out of the change. To distinguish them from rigourously implemented local transactional systems, we define two concepts that respect widespread beliefs about change at the level of systems:

Definition 45 (System rollout) *A distributed intentional change Δ implemented in a quasi-transactional manner. This may or may not involve the suspension of a system operations.*

Definition 46 (System rollback) *A distributed change Δ^{-1}, relative to an intentional change Δ, implemented in a quasi-transactional manner. This may or may not involve the suspension of a system operations.*

Transactional rollback is the inverse of an atomic 'transaction commit' operation, i.e. it is supposedly an atomic 'undo' operation[3].

The concepts defined above mimic the notion of transactional commit and rollback in locally isolated monitors, such as computer supervisors and database engines, but the usage does not satisfy the isolation requirements for proper transactional behaviour.

5.4.2 Rollback is impossible in an open system

The popularity of transactional thinking has led to a similar popularity of rollback thinking: when a mistake or unforeseen error is committed, one simply needs to roll back to the previous state. A formal proof has been given to show that this is impossible in general[BC11].

Version control systems use the same basic technique of approximate isolation of state with content branches, though it often goes awry, due to the realities of imperfect isolation. This is because changes have dependent consequences, arising from sampling the faulty states during the non-atomic changes. We cannot control time is a distributed system.

5.4.3 Phased rollout - risk mitigation processes

Limiting the scope of 'rollouts' can limit the risk of unintended side-effects, but it cannot predict all of them. Some problems might occur only when a certain scale is reached. Phase rollout effectively limits the amplification os possible fault by limting $|R|$ in equation 3.87, see the discussion of gain in section 3.7.1. However, what one cannot see from the instantaneous response/gain is the network effect of extended fault propagation.

Even if one could imagine transactionally committing a change, the unintended consequences may only be visible over an extended time, long after the transaction has completed. Such effects can only be estimated by understanding the timescales relating to dynamics of the system; this is difficult to predict when systems reach a high degree of complexity, such as near instability.. During that time interval, intermediate states can arise preventing the possibility of a rollback, even if it is still possible to recreated an approximation to transactional behaviour[BC11].

By limiting the scope, one increases the chances of finding instabilities, errors, and provoking faults in the larger system while still maintaining a partially working system but this is not guaranteed, as critical instablities could still be provoked. See section 4.12.2.

Our best chance of avoiding loss of quasi transactional properties is *fault tolerance*, which can absorb cracks before they propagate.

Validation of input

A/B testing ... backout

5.4.4 Cleanup after a failed rollout

Faults that have already propagated to other agents

[3]Text editors can do this as long as there is only a single journal of changes and a single user interacting with the data. Local databases can likewise maintain process integrity.

Transaction locking does not apply to all state in a distributed system, because there is no monitor of all resources (kernel, hypervisor, etc) that has jurisdiction over everything.

The time window in which a semi-complete transaction allows partial state to propagate.

Example 33 (Push/imposition and rollback) *A change is made to a firewall accidentally and erroneously shutting it down. A computer gets infected. The change is rolled back so the firewall is replaced, but the server is already infected, hence the fault persists.*

There is an obligation to remove your shoes before entering houses in Japan, Norway, etc. If you are unaware, you might unwittingly make a perceived error in following this imposition. To correct the error, you remove your shoes, but now you have to clean up the dirt you already walked into the house. Thus a rollback to the approved state does not undo the problems associated with erroneous push.

5.4.5 Rollout by push and pull

Changes that are *pushed* from a single agent are in an unknown state by the instigator until each agent being imposed upon reports its state. There is no global atomicity, and no plausible rollback (imposing state through a noisy channel with crosstalk is a low fidelity operation).

Changes that are *pulled* by a set of agents always maintain a known distributed state by the instigators. There is limited local atomicity, and limited chance of repair in case of local error. The distributed consequences of a correctly implemented local change might still be incorrect at a larger scale. In this case, the effect is emergent and cannot be rolled back.

5.4.6 Convergent repair or rollforward

5.5 System upgrades and maintenance

Do we take systems offline while upgrading? Loss of service. Transactional?

5.6 Interventions

Sometimes a system that is only quasi-stable or weakly unstable can operate from intervention to intervention. This is much how a jet fighter works.

This is how we handle chaos, weather events etc

Long tailed distributions and buffer sizes

5.7 Understanding system timescales

The dynamical behaviour of a system is both governed and characterized by a set of timescales.

The maintenance theorem [Bur03] states, in essence, that a fault arrival process can be corrected, or brought into a dynamic equilibrum with a maintenance counterprocess, provided the counterprocess works fast enough to correct the fault before they exceed the tolerance limits of the system.

Type I and type II theories

5.7.1 The game of maintenance - an arms race by rounds

A game between faults versus maintainers. Tit for tat tournement The language of game theory helps to look at this problem through a formal lens.

Epochs as game rounds

It is a game because it has changing semantics. It has many rounds because the conditions of the game are changing. We look for equilibrium, but the abilty to attain equilibrium depends on the timescales at work.

Arms races

coop games

Keeping the promises is a type II game[Bur04a]

Rate of keeping
Nyquist sampling rate

5.7.2 Steady course in stormy weather

To keep the same semantics, in a changing environment, we have to adapt.

5.7.3 System upgrades, releases, bugs and fixes

The motivations for system design upgrades

- Adjustments to promises due for fitness for purpose
- Adjustments to implementation algorithms for keeping promises.
- New promises to increase the coverage of maintenance and feedback.

5.7.4 Does the redundancy folk-theorem hold for interactions?

Does the redundancy folk theorem still hold in a promise network? We've already shown that it holds in the limit of perfect cooperation, but can we assume that it still holds in the case of other strategies, such as fault tolerance, etc?

(TO DO) Probably not, this needs to be investigated...

More modes of failure....

5.7.5 Prevention is best, but repair is likely

We will never be able to prevent faults and errors from occurring, especially regarding semantics. In software systems, there are just too many conflicting semantics in play, making mismatches of intent likely.

In industrialization, we work hard to eschew the semantics so that only mechanical repeatable behaviour can be maintained in a steady state.

Our best hope then is to be able to repair systems quickly enough that they will lead to catastrophic outcomes.

Safe failure modes.

Service levels should never be maintained as discrete states (e.g. binary on/off, working/broken), but scaled so that average behaviours are maintained.

The rate of decay of service levels should be less than the rate of repair. This is the maintenance theorem.

Don't wait for a fault that is critical. Maintain equilibria.

Rollback is not a recovery method.

5.7.6 Quick repair is indistinguishable from avoidance

Nyquist

Comment 25 (Quick repair is the only way to handle unpredictable outcomes) *Because what you can't foresee, you can't promise.. so this must be part of a longer term game.*

This might not just be about keeping the existing set of promises, because that set might not be an adequate envelope for containing the system's behaviours.

5.7.7 The problem with patching

The loss of repeatability is delta change management is a huge problem.

If you are trying to get to London, a sequences of drive 100 yards and turn left instructions is a lot less helpful than simple "follow the signs for London". Because the information is compressed and contextual rather than extensive and context free.

Comment 26 (Change is not only intended) *During forensic analysis of problems, it is often assumed that changes occur in a system by deliberate intentional action. This is not always the case. Both intended and intended changes can be instigated by agents. Moreover, changed that come from environmental impositions can also affect systems, particularly when they have not been designed to maintain their state in the face of such changes.*

5.8 Can agents be repaired or replaced to improve reliability?

5.8.1 Human fidelity within a system

The ability for humans to keep promises, depends on many factors. As agents in the system, humans may fail to work reliably relative to a situation or process, for a variety of reasons, some related to individual condition, and others to context:

- Management errors.
- Forgetfulness/carelessness.
- Misunderstanding/miscommunication.
- Confusion/stress/intoxication.
- Ignorance.
- Personal conflict.
- Slowness of response.
- Random or systematics procedural errors.
- Inability to deal with complexity.
- Inability to cooperate with others.

In system administration the problems are partly social and partly due to the cooperative nature of the many interaction software components. The unpredictability of systems is dominated by these issues.

Humans filter all communication through their own view of the world. We respond to things that make sense to us, and we tend to reject things that do not. This can lead to misunderstanding, or only partial understanding of a communicated message. It can be modelled as the projection of a signal into a digital alphabet that describes our limited domain of understanding. We match input to the closest concept we already know.

Unlike machines, humans do not generally use reliable protocols for making themselves understood (except perhaps in military operations). A system administrator or user can easily misunderstand an instruction, or misdiagnose a problem.

5.8.2 Agent anomalies: machines and humans

Regardless of their role in a system, we can study what issues cause agents to act erratically, or with low process fidelity. Some characteristics are particular to the type of agent. Here are two ad hoc categories, commonly discussed:

- *Machines* tend to have relatively few parts. They are relatively simple, rigid and are therefore easy to stabilize/control with static or locking equilibria.

  Machines do not act with moral safeguards that humans take for granted, thus a machine would dispassionately kill a human being or bring down an entire system in order to keep a promise. Thus we can take less for granted when agents are working like machines.

- *Humans* have many moving parts that achieve balance through dynamic equilibria. Holding an arm up requires constant firing of muscle contractions, there is no locking mechanism.

  Humans generally need to perceive meaning in what they do, else they will notcare about the outcome. Meaningless repetition is worse than meaningful repetition, with bespoke adaptation. There does not have to be a high degree of creativity involved, as long as there is a sense of fulfilling a purpose that matters to a person.

Of course, humans are also highly complex biological machines, so these are not unrelated categories: stereotypes are commonly misunderstood. In industrialization, we often ask humans to act as machines, suppressing thought and judgement for speed and efficiency. This leads to a mismatch of expectations.

Comment 27 (Discipline and constraint) *Process rigidity is classical management/control thinking, e.g. disciplining humans through forms, procedures and protocols. Machines or tools that apply strong constraints or safeguards are another example of discipline.*

In engineering we know that certain tools are fit for purpose, and others not so much. Asking humans to behave like machines may ultimately not help, in the long run, because imposed constraints are not self-motivated. Humans find ways around such mechanisms[a].

[a] This suggests that use of simple user interfaces may not increase human fidelity in difficult circumstances.

Repetitive work does not suit workers with low accuracy or fidelity, even with formulaic work, as repetition amplifies the probability estimator of error. Templates may be rendered incorrectly, with random errors, for example a simple probability of error can be amplified by application in batch $N(t)$, or by repetition over time T, giving a cumulative error of the form:

$$N_{\rm err} = \frac{1}{T}\int_0^T dt\ p_{\rm err}(t)N(t) \tag{5.3}$$

Thus, when dealing with agents whose fidelity is low, we can mitigate the errors and faults in a number of ways:

- Since the sum/integrand are purely positive, keep the working duration T short to minimize the number of errors.
- Reduce the probability of error $p_{\rm err}(t)$, and ensure that this is not an increasing function of time, due to fatigue, etc.
- Reduce the number of actions per unit time $N(t)$, to limit the exposure.

A simple way to solve all of these issues it to take the low fidelity agent out of the loop. The worst possible approach would be to connect a low fidelity agent to powerful amplifying automation.

5.8.3 Feedback and testing

- Remove positive feedback, and introduce negative feedback.
- Remove instabilities and non-linearities.
- Try testing at large N to detect flaws before production use (acceptance testing).

5.8.4 Blame, replacement, and agent error

The emotional reaction to loss, when a fault occurs or an error is made, often fuels a need to lash out. When accidents and disasters occur, it has been common for survivors to initiate lawsuits and seek to blame scapegoats. People kick their machinery when it fails to meet expectations. Human error is often the attributed cause. Humans are an easy target. If we take any thread of cause and effect within a system, there will always be a human at the end of it, because humans are the source of all intent. Thus looking for a scapegoat often becomes a case of 'pick a card, any card' to make a pain go away like a magic trick.

One thing in humans' favour is their capacity to learn. If a person makes a mistake, they will likely not make the same mistake again any time soon, whereas a replacement (who did not learn the same lesson) easily could. Thus replacing a learning agent that commits an error is not a rational response. Replacing a faulty part that is incapable of learning might be rational if one knows that the same promise is likely to be broken again.

What we see is that, in the search for emotional catharsis, we can make matters worse by increasing the likelihood of new faults.

5.8.5 Can we replace agents for better reliability?

A traditional argument for eliminating errors from a system, especially human errors, is to replace human workers with machinery, or to transform humans into procedural machines. As already mentioned, the culture of incomplete system design tends to result in the diagnosis of 'human error', when we have been too lazy to design fault tolerances and contingency promises. This suggests that simply replacing humans with machines might reduce errors of execution, increase speed and thus perhaps reduce faults, but cannot address design flaws, which embrace both of the latter.

Paradoxically, the more we deploy automation within human-computer systems, the more skilled human involvement we need to plan and guide the collaborative processes, and interpret the contextual learning.

Humans, on the other hand, are certainly good at learning and improving the coverage of promises, as a work in progress, and thus they have a natural skill within a system for *continuous improvement*.

This points to the fallacy of looking at the nature of the agents themselves, rather than focusing on whether or not they keep their promises. The most unlikely agents can sometimes do the best job[4].

5.9 Strategies for robust design (without flaws)

(TO DO)

Assuming that the arrangement of components in a system is determined by its function, first and foremost, let's ask what measures we can take to reduce unreliability in the agents.

5.9.1 Detailed promise design

1. What agent makes the promise? (Human/Mechanism?)
2. Why?
3. In what context does the promise apply?
4. Is the agent's promise adaptive or fixed/dumb with respect to context?
5. Who are the stakeholders?
6. Details of the body

5.9.2 Convergence: Fault tolerance under amplification

Divergence Idempotence/convergence - key to tolerance

5.9.3 The phase averaging trick for noise reduction

If we can introduce channels that absorb errors with the opposite sign, then they can be cancelled on recombination. This is difficult for semantic errors, but easier for dynamical errors.

This is the trick used in making balanced line audio cables.

[4]The example of using maggots to clean a wound comes to mind. We associate maggots with the disgusting dirty flies they become, yet maggots eat only dead flesh and discriminate far more efficiently than any artificial mechanism we have yet created.

5.10 Summary of promise keeping strategies

(TO DO)

Because there is a competition between errors and agents and methods are discrete, the strategies essentially form an array of alternatives that can be mixed like a strategic game[Bur04a]. This

Mixed strategies

coop games

Keeping the promises is a type II game[Bur04a]

Rate of keeping

Nyquist sampling rate

Repair within the sample interval - or failover within sample interval

This gives us some requirements for the nature of a promise body in a reliable system.

Semantics: details complete

Dynamics: timing sufficient to meet promise expectations (-).

Equilibrium, fault tolerance, caching Rate of promise assessment and sampling must be specified.

Avoid middlemen, as thse reduce the reliability (Schwarz?)

What kind of promises should agents make in order to support expectations?

Appendix A

Emperical examples: cases and remedies

> "Skilled workers tend to think that it is enough to be smart. In fact this is wrong: smart people tend to be problem solvers and will happily solve the same problem many times, wasting time and effort. Moreover, human intervention is often based on panic and lack of understanding so every time someone logs onto a system by hand, they jeopardize everyone's understanding of the system. Only the self-discipline of stable procedures leads to predictability. "
>
> –The Author, CFEngine 3 tutorial (2010)

The following is a selection of faults observed in the field for service providers, in boldface. Proposed resolutions, based on a promise-compliant architecture are described for each.

A.1 Observed faults and their avoidance

A common theme in these examples is the misplacement of trust in low fidelity agents. Buffers and noise reduction techniques (dynamic and semantic averaging) may be employed to reduce the risk of faults.

A.1.1 Key parameters were unexpectedly modified

Examples:

1. **Some switch parameters which should be OPEN are CLOSED, which was not realized until service crashed. e.g. routine backup, MAC self-learning function, etc.**

   No documented promise was made to provide these services, so no assessment of the state was made either, meaning that the state of the switch parameters are ad hoc, and not automatically repairable.

   This is common in models based on change, instead of desired outcome/state. (We make models about how we can get involved, not about what outcomes we desire.)

   By using a policy model, and a self-repairing agent with regular maintenance (e.g. CFEngine), because we cannot guarantee no tampering, this kind of error can be eliminated within a minimum time window. e.g., if an agent checks the state every T minutes, then the Mean Time To Repair (MTTR) is $T/2$.

2. **Some global parameters were modified by mistake and affected large scale services.**

   Strict routines for non-tampering with devices should be introduced. A barrier or buffer between humans and device states is needed so that human whims and spurious actions are isolated from high risk outcomes. The risk is higher when instructions and changes are partial or relative (context-free) changes, rather than complete repeatable outcomes. The loss of repeatability is a key problem.

   For a software system, all changes may be version controlled changes to a context aware policy that describes a clear outcome for each context. The policy is then implemented and maintained by a self-repairing agent with convergent outcome.

Comment 28 (Model based promise-making) *The key principle that is violated in these examples is that a user or consumer of a service has clear expectations of a promise being kept, but no promise was actually made: ad hoc configuration changes were once by someone, without verification.*

By employing a model-based promise language, a promise can be made a kept formally and verifiably. Changes can be versioned and tracked, pre-tested, etc. This is a standard approach used for:

- *Compliance with regulations*
- *Compliance with Service Level Agreements*

A.1.2 Wrong modifications applied in batch

Examples:

1. **A user wants to modify the LAC attribute of a single base station[1], but actually all LAC of base stations was modified, because the user select the ALL parameters by mistake.**

   Inadequate buffer between human low-fidelity agent and amplified outcome. A high-impact control parameter has inadequate safeguards to accidental change. e.g. the pilot ejecter seat on a plane should not be located next to the intercom radio button. A small mistake could lead to catastrophic consequences.

   A noice reduction method is needed, based on semantic averaging. For all amplified or catastrophic changes, implemented by low-fidelity agents, there should be multiple opinions, e.g. 'dual key' launch codes, or confirmation from a second (multiple) source. This is why pilots have co-pilots, missiles have dual launch keys, etc.

2. **A user wants to disable a device in a tree model network, but selects the upper level node by mistake, so all the devices under this node are been disabled.**

   The user is implementing actions directly instead of decoupling reasoning from action. Reasoning should be slow, and deployment fast only after validation. No validatable promise was made about what devices should be active, or the agent's assessment of the state was erroneous, perhaps due to insufficient fidelity or resolution in its assessment of the context.

   Again, this is a design error of placing radically different things close to one another (control channel separation), and trusting an unreliable agent without asking for confirmation? A warning message is the minimum barrier to a fault: "warning you are about to delete ... Are you sure?" A two-factor authentication with one-time pad codes can be used to authorize such events. This is common in online banking. For high risk items, dual keys can be required.

   This could also be dealt with by fault tolerance, ensuring that an erroneous change did not have major consequences, or requiring dual-key activation.

Comment 29 (Big hammers need supervision) *The key principle violated here is in coupling high impact consequences to a low fidelity agent, without verification. Spurious choices are thus amplified with high fidelity by automation (see section 3.7.3), This is like giving a child a sledgehammer to open a bottle.*

Role Based Access Control (RBAC) can set limits on what individual operators are allowed to do, with or without confirmation. Some operations are harmless and one can trust even a low-fidelity agent to implement them. When destructive operations are implemented with a high degree of amplification, but high fidelity, semantic averaging can be used (dual authorization).

As part of a version controlled language of promises, changes can be made by using a version control system (e.g. git), and filing a change request. A process supervisor then merges to offer confirmation. The code can be checked for consistency. Provided the workload on the supervisor does not lead to low fidelity verification, this is usually good enough.

[1] 3GPP Technical Specification TS 23.003 Numbering, addressing and identification contains a section defining the Identification of location areas and base stations, using LAI, LAC, RAI, RAC, CI, CGI, BSIC, RSZI, LN, SAI. Also a section on Identification of mobile subscribers, using IMSI, TMSI, P-TMSI, LMSI, TLLI.

A.1.3 Deletion of key resources by mistake

More examples of inadequate separation of (potentially) low fidelity human agent from a dangerous outcome (misplaced trust):

1. **Deleting a route by mistake.**
2. **Delete a IP path by mistake.**
3. **Some configuration files deleted by mistake.**

   No manual change allowed. Model based change with version control is used, and the "diff" is authorized by duplicate sources of truth before being deployed.
4. **The user deletes the wrong resources, or he did not realize there are key services on these resources.**

   No manual change allowed. Model based change with version control is used, and the "diff" is authorized by duplicate sources of truth before being deployed. If resources are in use when deleted, this can be detected from the model documentation. Knowledge-based tools and keep track of these.

Comment 30 (Dependence tracking and change impact) *In a system, any change can have consequences. A map of knowledge about these dependences can be inferred from a promise model.*

A key question to ask is: why would a user end up in this situation of wanting to delete a resource? What leads to this decision? Then, what context information does he/she need to make the decision?

A.1.4 Parameters or attribution are wrongly configured

Examples:

1. **The attribute of a port has been wrongly configured. It should be an optical interface, but was configured as an electrical port.**

   Low fidelity agent makes an erroneous assessement of current state, or erroneous response. Replace the agent/promise with a higher fidelity agent/promise.

   This is an error of understanding or intent. This could be due to inexperience of the human operator, or bad naming conventions. Clear naming conventions allow different components to promise their functions clearly. It should not be possible to confuse one type of component for another.
2. **An IP address been wrongly configured, so the service is directed to the wrong destination.**
3. **The ID of a device is wrongly configured.**
4. **The route map table is wrongly configured.**

   These are errors or intent. The lack of a proper model of the system, verfied and pre-tested, and rolled into production without testing. Again, clear naming is important for human comprehension and recognition.

   If the configuration errors are made in the policy model itself, this suggests a lack of understanding of the system, and a lack of testing.

Comment 31 (Proper identification through classification and naming) *The fidelity of human agents depends on their ability to make sense of a situation. Clearly distinguishable names and identities aid human comprehension.*

When systems process too much raw information, this leads to delays, fatigue, and loss of human comprehension. By using patterns to abstract/represent many things as a single pattern, we can make generic rules or wildcards to amplify the effect of a smal amount of information (see earlier remarks about amplification).

A.1.5 Configuration of key resources or parameters is missing

Examples:

1. **APS group 1022 of 106 did not get deployed, so a service packet did not send at both working tunnel and protection tunnel, so the protection tunnel lost function.**
2. **The VLAN was not divided and both two NEs connected to one maintenance LSW cause the issue and service crash.**
3. **Forgot to change IP after software upgrade, service abnormal.**

These are all examples of a lack of clear promises, implemented by high fidelity agents.

A.1.6 Inconsistent configuration

Examples:

1. **The configuration on master and slave board are not consistent, the configuration on slave board is wrong. But this mistake can only be found when the switch from master to slave is happen. At this time, the service is affected.**
2. **The docking parameters between two devices are configured inconsistently. Such as frame format, Name, policy of two devices are not consistent of conflict.**

These issues can be dealt with by model-based promised policy. Fault tolerance of system inconsistency is the most important property of a system. Full consistency cannot be guaranteed at all times. It is affected by network partitions and latencies.

A.1.7 Non-Optimal configuration

Examples:

1. **Parameter is too small or too big (but still in valid interval)**
2. **Some parameters are configured too small, its still in valid interval but have high risk. In the condition of certain business volume, it will trigger incident.**
3. **The network is configured in a non-optimal way.**
4. **VLAN is configured and split in non-optimal planning, so the service quality is affected.**

Optimization requires an understanding of how causal factors lead to desired outcomes. This can be learned over time by human analysis, by collecting data. For this, we have to combine intended state with actual state and output.

A.2 Challenges

Examples:

1. **A mistake has been introduced into the network long ago, but was undetected until the service was seriously affected, leading to many incidents.**

   Manual tampering must be forbidden. Errors in the model can be verified by acceptance testing, or oversight.

2. **A wrong configuration causes bulk parameters or data be modified. Sometimes the error can be found immediately, but it will take too long to change the parameters or data back to the initial status, leading to long term service disruption.**

   Agent based automation, based on policy models minimized this risk. Errors to policy itself can be addressed by testing, or consistency checklists. The move towards disposable container technologies allows quick reversal of configuration in stateless systems. A dependence on runtime state leads to fragility here.

3. **An error been introduced into the network, because of the lack of complete validation rules. But it is a huge work to work through the code to find which is lacking.**

   This is an intentional error. Validation is simpler in a promise model.

4. **Configuration errors take too long to discover.**

   Regular automated verification should be run at regular intervals that matches the likely rate of fault (MTBF).

5. **There are different vendor products, and some configuration is associated with vendor product, but some with service type. Its hard to find a solution or technology to help all products to resolve the problems.**

   A multi-vendor (open source) agent is straightforward. This is the approach used in CFEngine across dozens of different systems and vendors systems.

A.3 Common datacentre failure modes

1. Device failures - bad batch of disks, PSUs
2. CPU failures - cache corruption, math errors
3. Datacentre failures - power network, disaster
4. Routing failures - DNS, Internet/ISP path
5. Vendor diversity

A.4 Software failures

1. Time bombs - counter wrap, memory leak
2. Date bombs - leap years, seconds, epochs etc
3. Expiry - certificates expire
4. Revocation of credentials - Certificate authority failure
5. Security exploit
6. Language bugs - compile time
7. Runtime bugs - JVM, Linux, Hypervisor
8. Network bugs - routers, firewalls, protocols
9. Versioning mismatches
10. Configuration errors
11. Overload
12. Build was interrupted or failed to complete
13. Timeout

Semantic averaging: Cultural diversity leads to an effective mixed strategy and mutuation exploration

A.5 Fabric failures

The promise that pulls to resolve this push conflict is a typically a lock server. But aggregators can also resolve this, like CDN.

A.6 Predicting new failure modes at scale

Certain unpredictable occurrences occur at scale

What if the ‘weather’of rapid scheduleing of resources becomes so heavy that it begins to create effective couplings though third parties? (see section 3 body coupling - intermediaries with implicit causation)

Second order effects become more likely, amplified at scale

“black swan”

What symptoms might be exhibitied

A.7 Can we stabilize systems without breaking them further?

Suppose we design a system set of promises and assumptions. This leads to an unexpected outcome. Is it possible to say anything about how to stabilize the outcomes so that we could make/keep new promises to constrain the behaviour, without destabilizing the existing promises?

Modularity is a common assumption here. If two parts of a system are not in contact, them how can they affect one another. However, this is naive, as it ignores first and second order interactions. It is the argyment of classical component reliability.

This is the software bug fixing problem.

Appendix B

Summary of *do* and *don't* in system design

Can we boil down all this tough study into some rules of thumb? Should reference these to the text....

We summarize the analysis of fragilities with respect to promise keeping from the foregoing sections. This describes a mixture of general suggestions for stability of both dynamics and semantics.

Learning (so-called 'anti-fragile') systems become increasingly 'situation aware', such that predictability and robustness can grow. We avoid designs that rely on memory or suffer from system memory loss, such as unstable, and non-linear designs that lead to mixing and loss of predictability.

B.1 Fault related principles

1. DO Reduce fault surface of system $|in\rangle$.
2. DO Reduce amplification of effect $\langle out|in\rangle$.
3. DO Reduce degrees of freedom with promised constraints.
4. DO Equilibrate degrees of freedom.
5. DO Increase sampling rate for fault detection, Nyquist frequency.
6. DO Deccrease MTTR Δt.
7. DO add preventative processes to increase MTBF (increase tolerance, or slow system velocity).

B.1.1 Devices (machine proxy agents)

1. DO Avoid operations that apply relative changes (deltas). Make all operations converge (idempotently) to a fixed desired end-state.
2. DO Make all behaviours fault-tolerant of their own states, and of dependencies. Every operation should have an outcome that is aligned with goals, despite unexpected conditions.
3. DO Make state self-maintaining.
4. DO make devices self-recovering in case of fault.
5. DON'T rely on alarms and human intervention in critical situations (see comment 20). DO pre-plan for disaster scenarios and build in safe rational responses that converge towards a desired end-state.
6. DO Avoid hard dependencies. Cache/store all pre-requisites at the point of requirement before acting to prevent unpredictable failure.
7. DON'T allow any state of the system to block a control channel to an agent (human or device).
8. DON'T (ever) assume that all devices are synchronized or in a consistent state.

9. DON'T assume availability of any device.
10. DO make agent models. DON'T make agentless models. DO apply repairs and controls at the fault location, never by remote control (because this compounds uncertainty).
11. DO ensure that all devices and systems fail safely and predictably.
12. DO consider the need for redundancy and DO evaluate the need for multiple opinions and quorum (semantic averaging). It is most stable to be tolerant of all variation in a system.

B.1.2 Human agents

1. DO assess all promise proposals, in the light of all possible contexts, before making them.
2. DO review promises regularly (continuous assessment and delivery) and consider possible missing promise coverage.
3. DO change promise intent slowly, but contextual details continuously, with version control, and test vaidation, to trace who changed them and why.
4. DON'T use transactional thinking. Don't pretend that something is transactional if it is non-atomic and without proper isolation. Few system scenarios fit these conditions.
5. DON'T allow any state of the system to block a control channel to an agent (human or device).
6. DO consider the need for redundancy and DO evaluate the need for multiple opinions and quorum (semantic averaging). It is most stable to be tolerant of all variation in a system.

B.1.3 Human-machine interaction

1. DO limit the focus of any interface to a clear goal to increase situation awareness.
2. DON'T provide direct (CLI or GUI) access to mission critical functions. Use policy to buffer changes through an intermediary (with authorization, verification and validation).
3. DO require confirmation of intent by compiling intentions and validating them before execution.
4. DO provide graded alarms as advisories. DON'T plan to rely on new reasoning in a crisis (see comment 20).
5. DO avoid relying on specific ordering of events or promises being kept. Fault tolerance of ordering and preconditions will avoid a major source of brittleness.

Direct action is always more risky than buffering decisions through a proxy. The aim is to allow ourselves time to evaluate reasoning before executing or acting on the decision. We want to avoid risky situations, where rapid decisions insert new instabilities. Decisions on the timescale of changing environmental events leads to non-linear coupling, and hence increases the likelihood of instability. A good design goal is to separate slow reasoning from fast pre-determined response.

B.2 The value of promises

The value of links in a network depends on the promises they make. The value of a promise is a form of assessment[BB14] that any agent can make independently. We write an assessment of whether a promise was kept

$$\alpha_i(A_i \xrightarrow{b} A_k) \in [0, 1] \tag{B.1}$$

to mean the assessment by agent A_i that the promise from A_j to A_k was kept. A valuation is an estimate of what a promise is worth to an agent. This may of may not depend on the assessment of to what extent the promise is kept

or not. Every agent assesses on its own calibrated scale. If we want a common currency valuation for all parties, this has to be calibrated by a single agent according to its scale.

The interpretation of value is also an individual judgement that relies on trust, and may be based on accounting of the assessments over time (reputation)[BB06].

$$\text{My reputation} \propto \sum_{\text{you}} \left(\text{you} \xrightarrow{-b} \text{me} \right) \tag{B.2}$$

In words, my reputation is proportional to all the number of 'you' who (publicly) promise to accept my promised service. Even unilateral promises may have some value:

VALUATION BY X	ABOUT PROMISE	REASON FOR VALUE TO X
Me	me $\xrightarrow{+b}$ you	An reputation building investment
Me	you $\xrightarrow{+b}$ me	A service that might help me
You	me $\xrightarrow{+b}$ you	A service that might help you
You	you $\xrightarrow{-b}$ me	You need the service now

Cooperative relationships are usually based on conditional assistance[BB14], and take the form of a conditional equilibrium:

$$S \xrightarrow{+S|M} R \tag{B.3}$$
$$R \xrightarrow{+M|S} S \tag{B.4}$$
$$S \xrightarrow{-M} R \tag{B.5}$$
$$R \xrightarrow{-S} S. \tag{B.6}$$

in words, S promises R a service, if it receives payment M; and R promises to pay M if it receives service S. In a network without trust, this is a deadlock. But if any agent trusts the other enough to go first, it is a cyclic generator of a long term relationship. Such relationships imply lasting value, as known from game theory (for a review see [Bur13]). Both agents also promise that they will take (-) what the other is offering unconditionally. This is a signal of trust. Valuations are not necessarily rational to anyone but the agent that makes them, and are unrelated to cost.

The economic value is that something is exchanged, which requires a binding of both + and - promises.

$$S \xrightarrow{+S} R \tag{B.7}$$
$$R \xrightarrow{-S} S \tag{B.8}$$

Both agents recognize the value of the other party, so the value exchanged is proportional their assessments that the promises were kept:

$$v_S \propto \alpha_S(S \xrightarrow{+S} R)\alpha_S(R \xrightarrow{-S} S) \tag{B.9}$$
$$v_R \propto \alpha_R(S \xrightarrow{+S} R)\alpha_R(R \xrightarrow{-S} S). \tag{B.10}$$

In a community where such transfers are made often and between arbitrary pairs of agents, standards of valuation are equilibrated, and may be exchange in league with a calibration agency (e.g. a bank or government). Thus, in a well-connected community, with a spanning infrastructure, we may posit that the value of a one-way transfer is simply

$$v_C(\Pi_{ij}^S) = c_S \alpha_i \alpha_j, \tag{B.11}$$

where c_S is the currency value of a perfect service relationship S, and α_i is an impartial assessment of the probability with which A_i will keep its promise to give or receive S.

Bibliography

[AHF+14] E. Arcaute, E. Hatna, P. Ferguson, H. Youn, A. Johansson, and M. Batty. Constructing cities, deconstructing scaling laws. *Journal of The Royal Society Interface*, 12(102), 2014.

[ALB99] H. Abdu, H. Lutfiya, and M. Bauer. A model for adaptive monitoring configurations. *Proceedings of the VI IFIP/IEEE IM conference on network management*, page 371, 1999.

[Amd67] G.M. Amdahl. Validity of a the single processor approach to achieving large scale computer capabilities. In *Proceedings of the AFTPS Spring Joint Computer Conference*, 1967.

[Axe97] R. Axelrod. *The Complexity of Cooperation: Agent-based Models of Competition and Collaboration*. Princeton Studies in Complexity, Princeton, 1997.

[Bar96] G.I. Barenblatt. *Scaling, self-similarity, and intermediate asymptotics*. Cambridge, 1996.

[Bar03] G.I. Barenblatt. *Scaling*. Cambridge, 2003.

[BB05] M. Burgess and K. Begnum. Voluntary cooperation in a pervasive computing environment. *Proceedings of the Nineteenth Systems Administration Conference (LISA XIX) (USENIX Association: Berkeley, CA)*, page 143, 2005.

[BB06] J.A. Bergstra and M. Burgess. Local and global trust based on the concept of promises. Technical report, arXiv.org/abs/0912.4637 [cs.MA], 2006.

[BB08] R. Badonnel and M. Burgess. Service load balancing with autonomic servers: Reversing the decision making process. In *Resilient Networks and Services, Second International Conference on Autonomous Infrastructure, Management and Security, AIMS 2008, Bremen, Germany, July 1-3, 2008, Proceedings*, pages 92–104, 2008.

[BB14] J.A. Bergstra and M. Burgess. *Promise Theory: Principles and Applications*. $\chi tAxis$ Press, 2014.

[BBCEM10] J. Bjelland, M. Burgess, G. Canright, and K. Eng-Monsen. Eigenvectors of directed graphs and importance scores: dominance, t-rank, and sink remedies. *Data Mining and Knowledge Discovery*, 20(1):98–151, 2010.

[BC03] M. Burgess and G. Canright. Scalability of peer configuration management in partially reliable and ad hoc networks. *Proceedings of the VIII IFIP/IEEE IM conference on network management*, page 293, 2003.

[BC04] M. Burgess and G. Canright. Scaling behaviour of peer configuration in logically ad hoc networks. *IEEE eTransactions on Network and Service Management*, 1:1, 2004.

[BC11] M. Burgess and A. Couch. On system rollback and totalised fields an algebraic approach to system change. *Journal of Logic and Algebraic Programming*, 80:427–443, 2011.

[BCE04] M. Burgess, G. Canright, and K. Engø. Inportance-ranking functions from the eigenvectors of directed graphs. *Journal of the ACM (Submitted)*, 2004.

[BD07] M. Burgess and M. Disney. Understanding scalability in network aggregation with continuous monitoring. In *Lecture Notes on Computer Science, Proc. 18th IFIP/IEEE Distributed Systems: Operations and Management (DSOM 2007)*, volume (submitted). Springer, 2007.

[Bet13] L.M.A. Bettencourt. The origins of scaling in cities (with supplements). *Science*, 340:1438–1441, 2013.

[BF07] M. Burgess and S. Fagernes. Laws of systemic organization and collective behaviour in ensembles. In *Proceedings of MACE 2007*, volume 6 of *Multicon Lecture Notes*. Multicon Verlag, 2007.

[BF08] M. Burgess and S. Fagernes. Laws of human-computer behaviour and collective organization. *submitted to the IEEE Journal of Network and Service Management*, 2008.

[BHRS01] M. Burgess, H. Haugerud, T. Reitan, and S. Straumsnes. Measuring host normality. *ACM Transactions on Computing Systems*, 20:125–160, 2001.

[BLH+07] L.M.A. Bettencourt, J. Lobo, D. Helbing, C. Hühnert, and G.B. West. Growth, innovation, scaling and the pace of life in cities. *Proceedings of the National Academy of Sciences*, 104(107):7301–7306, 2007.

[Bri06] B. Briscoe. Metcalfe's law is wrong. *IEEE Spectrum*, 2006. http://spectrum.ieee.org/computing/networks/metcalfes-law-is-wrong (retrieved 1.1.2016).

[BU06] M. Burgess and G. Undheim. Predictable scaling behaviour in the data centre with multiple application servers. In *Lecture Notes on Computer Science, Proc. 17th IFIP/IEEE Distributed Systems: Operations and Management (DSOM 2006)*, volume 4269, pages 9–60. Springer, 2006.

[Bur95] M. Burgess. A site configuration engine. *Computing systems (MIT Press: Cambridge MA)*, 8:309, 1995.

[Bur03] M. Burgess. On the theory of system administration. *Science of Computer Programming*, 49:1, 2003.

[Bur04a] M. Burgess. *Analytical Network and System Administration — Managing Human-Computer Systems*. J. Wiley & Sons, Chichester, 2004.

[Bur04b] M. Burgess. Configurable immunity for evolving human-computer systems. *Science of Computer Programming*, 51:197, 2004.

[Bur04c] M. Burgess. Configurable immunity model of evolving configuration management. *Science of Computer Programming*, 51:197, 2004.

[Bur12] M. Burgess. Deconstructing the 'cap theorem' for cm and devops. *http://markburgess.org/blog_cap.html*, 2012.

[Bur13] M. Burgess. *In Search of Certainty: the science of our information infrastructure*. Xtaxis Press, 2013.

[Bur14] M. Burgess. Spacetimes with semantics (i). *http://arxiv.org/abs/1411.5563*, 2014.

[Bur15] M. Burgess. Spacetimes with semantics (ii). *http://arxiv.org/abs/1505.01716*, 2015.

[Bur16] M. Burgess. A promise theory approach to understanding resilience: faults, errors, and tolerance within systems. Technical report, Available at markburgess.org, 2015-2016.

[CB09] A. Couch and M. Burgess. Compass and direction in topic maps. *(Oslo University College preprint)*, 2009.

[Cha05] Robert N. Charette. Why software fails: We waste billions of dollars each year on entirely preventable mistakes. *IEEE Spectrum*, 2005. http://spectrum.ieee.org/computing/software/why-software-fails.

[Cil98] P. Cilliers. *Complexity and Postmodernism: Understanding Complex Systems*. Routledge, 1998.

[CT91] T.M. Cover and J.A. Thomas. *Elements of Information Theory*. (J.Wiley & Sons., New York), 1991.

[D̈89] D. Döner. *The Logic of Failure*. Basic books, New York, 1989.

[Dek06] S. Dekker. *The Field Guide to Understanding Human Error*. Ashgate Publishing, Surrey, 2006.

[Dek11] S. Dekker. *Drift Into Failure*. Ashgate Publishing, Surrey, 2011.

[DH06] D.D.Woods and E. Hollnagel. *Joint Cognitive Systems: Patterns in Cognitive Systems Engineering*. Taylor & Francis, New York, 2006.

[Die02] R.H. Dieck. *Measurement and Uncertainty (Methods and Applications)*. Instrument, Systems and Automation Society, Triangle Park, North Carolina, third edition edition, 2002.

[Dis07] M. Disney. Exploring patterns for scalability of network administration with topology constrants. Master's thesis, Oslo University College, 2007.

[Dun96] R. Dunbar. *Grooming, Gossip and the Evolution of Language*. Faber and Faber, London, 1996.

[GK00] P. Gupta and P.R. Kumar. The capacity of wireless networks. *IEEE Trans. Info. Theory*, 46(2):388–404, 2000.

[GPT15] N.J. Gunther, P. Puglia, and K. Tomasette. Hadoop superlinear scalability: The perpetual motion of parallel performance. *ACM Queue*, 13(5), 2015.

[Gun93] N. J. Gunther. A simple capacity model of massively parallel transaction systems. In *CMG National Conference*, 1993.

[Gun08] N.J. Gunther. A general theory of computational scalability based on rational functions. Technical report, arXiv:0808.1431, 2008.

[HF10] J. Humble and D. Farley. *Continuous Delivery: Reliable Software Releases Through Build, Test, and Deployment Automation*. Addison Wesley, New Jersey, 2010.

[HR94] A. Høyland and M. Rausand. *System Reliability Theory: Models and Statistical Methods*. J. Wiley & Sons, New York, 1994.

[HWL06] E. Hollnagel, D. Woods, and N. Leveson, editors. *Resilience Engineering*. Ashgate Publishing, Surrey, 2006.

[IEE] IEEE. A standard classification for software anomalies. *IEEE Computer Society Press, 1992*.

[LNP90] Kai Li, Jeffrey Naughton, and James Plank. Real-time concurrent checkpoint for parallel programs. In *Proceedings of the second ACM SIGPLAN Symposium on principles and practice of parallel programming*, pages 79–88. Association for Computing Machinery, 1990.

[Met95] B. Metcalfe. Metcalfe's law: A network becomes more valuable as it reaches more users. *Infoworld*, October 1995.

[Nat98] B. Natvig. *Pålitelighetsanalyse med teknologiske anvendelser*. University of Oslo Compendium, Oslo, Norway, 1998.

[NRC81] U.S. Nuclear Regulatory Commission NRC. *Fault Tree Handbook*. NUREG-0492, Springfield, 1981.

[OT06] A. Odlyzko and B. Tilly. A refutation of metcalfe's law and a better estimate for the value of networks and network interconnections. *IEEE Spectrum*, 2006.

[Rea90] J. Reason. *Human Error*. Cambridge University Press, Cambridge, 1990.

[SS03] M. Steinder and A. Sethi. A survey of fault localization techniques in computer networks. *Science of Computer Programming*, 53:165, 2003.

[SW49] C.E. Shannon and W. Weaver. *The mathematical theory of communication*. University of Illinois Press, Urbana, 1949.

[Tai88] J. Tainter. *The Collapse of Complex Societies*. Cambridge, 1988.

[Top72] J. Topping. *Errors of Observation and their Treatment*. Chapman and Hall, 1972.

[Wes99] G.B. West. The origin of universal scaling laws in biology. *Physica A*, 263:104–113, 1999.

[Wir95] N. Wirth. A plea for lean software. *Computer*, 28(2):64–68, 1995.

[ZLX15] X.Z. Zhang, J.J. Liu, and Z.W. Xu. Tencent and facebook data validate metcalfes law. *Journal of Computer Science and Technology*, 30(2):246–251, 2015.

[ZSHD04] W.X. Zhou, S. Sornette, R.A. Hill, and R.I.M. Dunbar. Discrete hierarchical organization of social group sizes. *Proc. Royal Soc.*, 272:439–444, 2004.