

Pervasive Computer Management: A Model of Network Policy with Local Autonomy

Mark Burgess and Siri Fagernes
Oslo University College, Norway

Abstract

We present a model for policy based management, stressing the role of *decisive autonomy* in generalized networks. The organization and consistency of agent cooperation is discussed within a cooperative network. We show that some simple rules can eliminate formal inconsistencies, allowing robust approximations to management. Using graph theoretical ranking methods, we evaluate also the probable consistency and robustness of cooperation in a network region. Our theory makes natural contact with social network models in building a theory of pervasive computing. We illustrate our model with a number of examples.

Index Terms

Configuration management, ad hoc networks, peer to peer, pervasive computing.

I. INTRODUCTION

The dream of pervasive computing is that of a future world in which computing devices are embedded into smart surfaces, kitchen appliances, houses, cities, and all manner of other commonplace objects. One of the central issues is how to manage such a vast number of devices, with so many different masters. Many organizations expect to be able to ‘control’ these devices using traditional management ideas. The question arises as to whether this is realistic.

Pervasion is not merely for smart environments. In addition to the ubiquitous devices, pervasive environments may be found in many other contemporary scenarios: data centres, web hotels, and even the autonomous agencies of the Border Gateway Protocol (BGP), where many ‘AS’s interact. It is about a common theme: the reliable interaction of many autonomous parts.

In this work we introduce a theory for describing pervasive computing management, beginning from its principal point of departure from classical management: autonomy. This introductory paper summarizes a framework for discussing the behaviour of policy driven, collaborative groups of computing devices. We confine ourselves to an overview of the model and its applications here, deferring detailed analyses and proofs for later work.

We study autonomy, not necessarily because it is ultimately desirable but because it is an extreme viewpoint, atomic in nature. Given autonomous building blocks, we can construct cooperative or competitive structures without prejudice. Previous approaches to model pervasive computing are mainly technology driven and take a data modelling view of a presupposed

architecture[1]. Ontology approaches and multi-agent system inspired work has been used[2]. A common theme has been to provide a framework, perhaps through middleware to map an adaptive environment onto a traditional computing network management scenario[3], [4], [5], [6]. Another theme has been the security model for pervasive computing. Again, the general approach is to try to design a security system with foreknowledge of the actors in a network.

Our paper differs from these approaches in that it presumes no prior design roles, nor any traditional system architecture. Rather it assumes or allows for completely autonomous service provision and uncontrolled emergent behaviour. The aim is not to suppress or design away such emergent behaviour, but to understand and manipulate it to the common good.

The outline of our paper is as follows:

- We begin by describing the idea of network policy using *promises*, a representation in terms of labelled graphs.
- We discuss some results and peculiarities of promises and the ideas of voluntary cooperation, and give an example of how this model applies to a network service view of management in a pervasive computing environment.
- Finally, we discuss simplifications of this theory, necessary in order to make the resulting picture manageable.

Our manifesto is to find a language for describing collaborative networks of basic services, such that their basic functions and roles are self-evident. It provides a recipe for approximating the complexity of a pervasive system to a level at which it becomes comprehensible and manageable. The attraction of our model is that it can synthesise many well-known results and therefore it makes contact with established areas of research in diverse fields[7], [8].

II. POLICY AND PROMISES

An important paradigm for controlled environmental adaptation is the *policy based management* scenario[9], [10], [11], [12], [13]. In this scenario, systems are configured by specifying constraints on their allowable states and behaviour, and these are propagated to them in the form of rule-sets[14], [15]. Each device is individually responsible for responding to events and changes within the bounds of these rules. Each system therefore becomes an autonomously deciding agent in a network collective.

We require a formal language for representing a specification of the host and network policy, both for definiteness, for comparison and for analysis. Several approaches have been used in the past[16], [17], [18], [19], [20], [21], [22]. Other formalisms that might be considered for describing policy include process algebras[23] such as the π -calculus[24] or Petri nets[25] or process algebras with probabilistic extensions[26]. However, these formalisms are more about understanding events in fully specified systems, with synchronization and resource constraints. They do not capture the behaviour of a system given an imperfect knowledge or specification, such as that one usually faces in a management scenario. There is thus additional uncertainty to be handled.

The language of voluntary cooperation is a language of promises[27] made in a *service* view-point of cooperation. Some promises are somewhat like Service Level Agreements (SLA)[28]. Autonomy implies that one cannot require anything of agents directly; all agents can do is to make promises about their own behaviour. In ref. [29], host policy is identified as a specification of the average configuration of the system over persistent times. An important aspect of this definition is that it allows for error tolerances, necessitated by randomly occurring events that corrupt policy in the system management loop. There is a probabilistic or stochastic element to system behaviour and hence policy can only be an average property in general. We do not require a full definition of host policy here from ref. [29]. It suffices to define the following.

Definition 1 (Agent node): An agent is an autonomous entity in a graph. Autonomy implies that each node makes all of its own decisions. No node in the graph can be coerced or may make a demand of any other. No agent can require any property of another.

Agents have knowledge and can perform (simple) computations. Promise theory is not related to the Actor Model[30], nor is the concept autonomous agents the same as term used in Multi-Agent systems[31], [32] where ‘autonomous’ simply means that agents work independently. We define a promise as follows:

Definition 2 (Promise): A labelled edge, $\langle n_1, \pi, n_2 \rangle$, in the graph whose nodes n_i are agents. We use the suggestive notation $n_1 \xrightarrow{\pi} n_2$ for the triple, to denote that n_1 asserts its compliance with a policy constraint π to n_2 .

The *promise body* π will in general consist of a *type* τ , which describes the domain of possibility, and a *constraint* χ which specifies the restriction $\chi \subseteq \tau$ imposed by policy. These might be

parameterized in further tuples. To avoid complication, we shall suppress the detailed structure of promise bodies in this paper.

A promise is a property of the two interacting nodes. For simplicity we shall think of promises as being timeless, i.e. we are suppressing the ability to have promise constraints depend on time in this paper.

Example 1 (Service Level Agreement (SLA)): As part of a Service Level Agreement, an agent X promises $\pi : \chi \subseteq \tau$ to agent Y , i.e. to provide service of type ‘database access in time q ’, τ is the type-domain $q \in [0, \infty]$ and the constraint $\chi(q) : 0 < q < 10\text{ms}$.

Definition 3 (Policy): A policy is simply a collection of promises.

Graphs are a natural and ideal representation of the relationships between nodes, but plain graphs alone are insufficiently expressive. Labelled graphs, at least, are needed to distinguish between the different kinds of inter-host relationships; an algebra of the labels is needed to evaluate the implications of combining relationships under local policy constraints.

Example 2: Some example promises (suppressing type) are suggested below:

- $X \xrightarrow{q \leq q_0} Y$: agent X promises to never exceed the limit $q \leq q_0$.
- $X \xrightarrow{q = q_0} X$: agent X promises to satisfy $q = q_0$.
- $X \xrightarrow{\ell \subseteq L} Y$: X promises to keep ℓ to a sub-language of the language L .
- $X \xrightarrow{S/P} Y$: agent X offers service S to Y conditionally if it receives promise P from some other agent.

Each variable above can be of a number of different types, relating to different services. All of these definitions have in common the simple set-theoretical form $\chi \subseteq \tau$.

As graphs between agents, promises have the form of a non-transitive, spatial Markov processes, since only nearest neighbours influence one another directly. If two neighbours develop a common set of values, in some sense, then one can make promises that are conditional on other promises. We can then form more general Markov chains, as we shall see below.

We shall identify three main kinds of promise below, which may be understood broadly as:

- Promises for giving (service) π .
- Promises for receiving (usage) $U(\pi)$.
- Promises for coordinating other promises $C(\pi)$.

The first two are fundamental, the third is a convenient construction, based on others.

- A *service promise* body, denoted by π , is the basic type of promise which denotes a restriction of behaviour by the promising agent in the manner of a service:

$$a \xrightarrow{\pi} b \quad (1)$$

involves an offer of knowledge from a to b and implies a specification of future behaviour towards b .

- A *usage or acceptance promise*, denoted $U(\pi)$, is the promise to receive or use a service π promised by another agent.

$$a \xrightarrow{U(\pi)} b \quad (2)$$

involves a receipt of knowledge by the promising agent a from b .

- A *coordination or subordination* promise, denoted $C(\pi)$, is the promise to do the same as another agent with respect to a promise body π . This is not a primitive type, but we denote it by the shorthand:

$$a \xrightarrow{C(\pi)} b \quad (3)$$

involves that b informs a about its actions with respect to promises of type π , and the receipt and usage of that information by a . This promise is a subordination for the agent because a is willingly giving up its autonomy in the matter of π by agreeing to follow b 's lead. We might call this a 'slave' promise.

The utility of promises is that they form a basic, graphical language for discussing the interactions of autonomous systems and their policies. The language is elementary and therefore requires some effort to use; however, a general abstraction with clear semantics allows one to build up systems according to easily understood patterns.

III. CONSISTENCY AND SUSTAINABILITY IN POLICY

There are many reasons why a policy promise might not be kept. These include unforeseen failures, human errors, specification errors, emergent effects, and so on. The breaking of a promise can be easily be viewed as a fault that is to be repaired. In this graphical framework, it is easy to make contact with classical reliability theory[33]. We shall only touch on this in the present paper.

If policy were simply a matter of finding a route from one node to another, as in network routing, one could provide multiple paths and other strategies for redundancy. However, the graph edges are labelled with constraints that must be compatible and hence there is a problem with the combinatorics of the constraints. One must consider therefore the effects of composing multiple constraints on the distributed system, allowing for the probabilistic nature of the individual assertions[34], [35], [36]. For the present work we assume a separation between the agents and the constraints which leads to an enormous simplification of logic:

Assumption 1 (Separation of constraint and identity): No promise constraints shall refer to the identities of agents in the graph.

One can make controlled deviations from this rule in order to generalize promises in a progressive manner. We shall make only one such exception in this paper in connection with coordination promises.

In some cases, the presence of multiple promises can lead to contradictions. Due to our assumptions of autonomy and separation, these are not severe. We call such contradictions broken promises, for want of a better term.

Definition 4 (Broken promise): A promise $X \xrightarrow{\pi_1} Y$ is said to be broken if there exists a counter-promise $X \xrightarrow{\pi_2} Y$, where $\pi_1 \neq \pi_2$.

Theorem 1 (Separation eliminates inconsistency): When the separation rule is obeyed, an autonomous agent can only break its own promises.

Proof: For a third party to be involved in the breaking of a promise, one must have transitivity of promises. Since agents are autonomous and have only knowledge of themselves, there can be no transitivity without referring to agent identities. ■

A promise might be deliberately intermittent in time, but that is not the same thing as it being broken.

A. Policy coordination

When autonomous agents make promises, they do so without any *a priori* regard for one another. Their promises do not have to be consistent, nor form a complementary whole. This is the price of autonomy. For management, we are interested in understanding what incentives it takes to make nodes behave cooperatively of their own volition. For this we must look at the

economics of the network; however, for the time being, we shall merely assume that cooperation is a desirable property between such agents and we seek a formal mechanism for binding nodes together without violating the assumption of autonomy.

Logically, the meaning of coordination is straightforward. Consider the three nodes in fig. 1. On the surface, a promise made by node b to node c does not imply any constraint or information

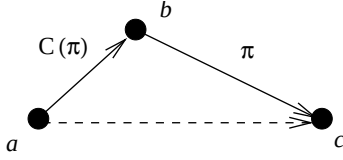


Fig. 1. Serial composition of promise and subordination, leads to limited transitivity.

about a promise from node a to node c : the nodes are autonomous. But, what if we introduce an additional promise from a to b ? Then, we can ask node a to coordinate with node b about its behaviour towards c . This requires an additional promise, but not the same kind of promise as b offers c . This must be a special type of promise: a meta-promise to b about the a new promise that a will make to c .

Definition 5 (Coordination (subordination) promise): Let $K(\pi)$ mean ‘knowledge about the promises with body π . We define the coordination promise:

$$X \xrightarrow{C(\pi)} Y \equiv \left\{ \begin{array}{l} Y \xrightarrow{K(\pi)} X \\ X \xrightarrow{U(K(\pi))} Y \\ X \xrightarrow{\text{Comply}(\pi)} Y \end{array} \right. \quad (4)$$

The slave agent promises to do the same as the master agent. This requires the master node to inform the slave of its intentions and for the slave to agree to comply with these.

Notice that this is also a subordination of the agent, since it promises to follow. It also breaks the ansatz about not referring to agent identities (in a controlled way). Hence we give this promise special status.

We may now write an equivalent promise for the case in which a promises b that it will coordinate on the matter of π , given that b promises c π . Let us use the symbol “ $\otimes$ ” to signify the composition of these promises.

$$\underbrace{a \xrightarrow{C(\pi)} b}_{\text{‘Coordinate with’}} \otimes \underbrace{b \xrightarrow{\pi} c}_{\text{Promise}} \Rightarrow a \xrightarrow{\pi} c. \quad (5)$$

This allows us to deduce something about the need for consistency of interaction with a third-party. We see that this combination of promises implies that there must be a promise π from a to c .

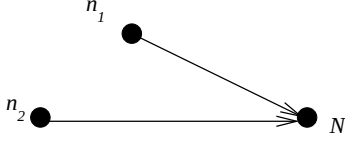


Fig. 2. Mutual coordination by third-party mediation.

Lemma 1 (Consistent promises): Two promises made to a third-party node, N , by two other nodes n_1 and n_2 , are consistent (see fig. 2), i.e.

$$n_1 \xrightarrow{\pi} N \Leftrightarrow n_2 \xrightarrow{\pi} N, \quad (6)$$

if and only if the two nodes additionally promise one another cooperation $C(\pi)$ in the matter of π .

Proof: Consistency implies

$$n_1 \xrightarrow{\pi} N \Leftrightarrow n_2 \xrightarrow{\pi} N, \quad (7)$$

but from, eqn. 5 we may write

$$n_1 \xrightarrow{\pi} N \Leftarrow n_1 \xrightarrow{C(\pi)} n_2 \otimes n_2 \xrightarrow{\pi} N \quad (8)$$

$$n_2 \xrightarrow{\pi} N \Leftarrow n_2 \xrightarrow{C(\pi)} n_1 \otimes n_1 \xrightarrow{\pi} N. \quad (9)$$

Thus, using eqn. (7) on both sides of these equations, we derive

$$n_2 \xrightarrow{C(\pi)} n_1 \Leftrightarrow n_1 \xrightarrow{C(\pi)} n_2. \quad (10)$$

■

By measuring p from both n_1 and n_2 , n_3 acts as a judge of their compliance with the mutual agreements between them (see fig. 3). This allows the basis of a theory of measurement or observation in collaborative networks.

Thus, for groups of agents to cooperate, what is required is a rule-based mechanism that allows independent agents to come together and form structures that span several nodes, by *voluntary*

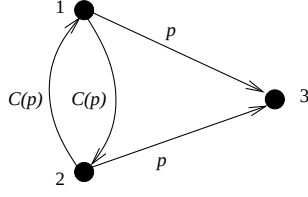


Fig. 3. Coordination and the use of third parties to measure the equivalence of agent-nodes in a region. Agents form groups and roles by agreeing to coordinate policy.

cooperation. Such an agreement has to be made between every pair of nodes involved in the cooperative structure. We summarize this with the following:

Notice, also, that if two nodes pledge their promise to an external party (a form of allegiance), then they have made an implicit agreement with one another also, which binds them together. This result is important for seeing the actual groupings of roles and structures in a policy graph.

Property 1 (Coordination promise rule): For two agents to guarantee the same promise, one requires a promise to coordinate with neighbouring agent-nodes.

A complete structure looks like this:

- n_1 promises p to n_3 .
- n_2 promises n_1 to collaborate about p (denote this as a promise $C(p)$).
- n_1 promises n_2 to collaborate about p (denote this as a promise $C(p)$).
- n_2 promises p to n_3

The cooperation promise will turn out to be an importance point of contact with the theory of games and voluntary cooperation[37], [7]; we return to this matter in a separate paper[38].

B. Common knowledge

Since agents' knowledge is private, they do not normally possess each others' knowledge about the world. The graphical construction assists us here. We assume that knowledge is transferred from an agent to another agent by virtue of promises, i.e. agents who send and receive promises are aware of them.

Theorem 2 (Common Knowledge): Let A be the set of agents and let $s \xrightarrow{\pi_K}$ be a service from a source agent s that provides information or knowledge K to other agents. The scope of knowledge K , originating from s , is limited to agents in the set of nodes that are 'downstream'

of s with respect to a promise π_K . The *guaranteed* scope of common knowledge K is limited to agents that are connected by a bilateral promise π_K that is accepted with a counter-promise $U(\pi_K)$. No other agent possess the knowledge of the source agent s .

Proof: Agents have knowledge only about themselves and knowledge is only transferred through promises with an acceptance handshake[39]. This theorem follows from the fact that the only nodes reachable by π_K promises are those connected to s by a used π_K promise (see ref. [40] for the importance of downstream nodes, and topological classifications of directed graphs). The π_K promise is necessary but not sufficient for dissemination. To *guarantee* the scope, we must have an additional agreement to use the knowledge provided, i.e. counter-promise $U(\pi_K)$ for each agent at distance i from s in the chain

$$\begin{array}{ccc} n_i & \xrightarrow{\pi_K} & n_{i+1} \\ n_{i+1} & \xrightarrow{U(\pi_K)} & n_i \end{array} \quad (11)$$

completes the construction for guaranteed knowledge. ■

C. Conditionals and dependencies

In order to reproduce familiar features of centralized management, such as cooperative dependencies, and requirements, we need introduce conditional promises and attendant agreements to obey them. To do this, we begin with the notion of a conditional promise (see fig 4):

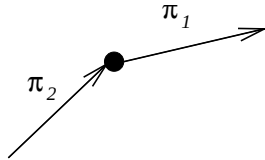


Fig. 4. A conditional promise is one promise that follows on from another.

Definition 6 (Conditional promise): A promise made conditionally on a Boolean condition C , known to the promising agent is written:

$$n_1 \xrightarrow{\pi/C} n_2 \quad (12)$$

i.e. n_1 promises n_2 π if the condition C is true.

Lemma 2: A conditional promise is not a promise unless the condition is also promised to be true (denoted $T(C)$).

$$\left. \begin{array}{l} n_1 \xrightarrow{\pi/C} n_2 \\ n_1 \xrightarrow{T(C)} n_2 \end{array} \right\} = n_1 \xrightarrow{\pi} n_2 \quad (13)$$

The proof of this is trivial from the knowledge of the agents.

There are potentially several ways of promising that C is true. e.g. I promise to deliver data if “I trust you”. One must then define a relationship between “I trust you” and other conditions. These relationships have to be established between promise types in advance, and lead to a grammar, which we shall not go into here. A simple construction we need a lot is the following. The receipt of a promise is defined to be ‘true’, for instance, so we write

$$n_1 \xrightarrow{\pi_1/\pi_2} n_2 \quad (14)$$

i.e. I promise you π_1 , if someone promises me π_2 .

$$\left. \begin{array}{l} n_1 \xrightarrow{\pi_1/\pi_2} n_2 \\ n_1 \xrightarrow{U(\pi_2)} n_2 \end{array} \right\} = n_1 \xrightarrow{\pi} n_2 \quad (15)$$

i.e. if I use π_2 then I can deliver on π_1 . This is one use of a use-promise.

Because each node is independent, a node that makes a conditional promise does not necessarily know about a promise that has been made to it. To make its conditional promise real, it must both be the recipient of a promise and confirm that it will use it by promising to accept the promised service. For typographic clarity we use the symbols $\otimes$ and $\oplus$ for serial and parallel composition of promises below.

Property 2 (Acceptance handshake): We define a third kind of promise, the “usage” or “acceptance” promise $U(\pi)$, and a notational simplification for the bilateral handshake agreement between two hosts.

$$\left(n_1 \xrightarrow{\pi} n_2 \otimes n_2 \xrightarrow{U(\pi)} n_1 \right) \Rightarrow n_1 \xleftrightarrow{\pi} n_2 . \quad (16)$$

The asymmetric arrow shows that n_1 is promising something to n_2 , and that n_2 accepts this, marking the fact that there is a qualitative asymmetry in the agreement: one node provides to the other, and the other merely confirms. One might want to call this construction an interface[41].

We now want to define what happens when we try to make a promise that depends on such an agreement. We consider the case of a forward confirmation of dependency, but the same argument applies for backward confirmation with use-promises.

Definition 7 (Dependency): A dependency promise requires the recipient of the service to receive assurance that the dependent service will be used.

$$n_1 \xrightarrow{\pi_d} n_2 \otimes \left(n_2 \xrightarrow{U(\pi_d)} n_3 \oplus n_2 \xrightarrow{\pi/\pi_d} n_3 \right) \Rightarrow n_2 \xrightarrow{\pi} n_3, \quad (17)$$

where $\oplus$ represents the parallel composition of promises. We use the following shorthand for this:

$$n_1 \xrightarrow{\pi_d} n_2 \otimes n_2 \xrightarrow{\pi/\pi_d} n_3 \Rightarrow n_2 \xrightarrow{\pi} n_3. \quad (18)$$

Example 3: For example, see fig. 5. Consider a web-server that depends on a database.

$$a \xrightarrow{\text{db}} b \otimes b \xrightarrow{\text{use-db}} c \oplus b \xrightarrow{\text{www/db}} c \Rightarrow b \xrightarrow{\text{www}} c. \quad (19)$$

The server b makes a conditional promise to the client c which is verified by the agreement about database support from a , and thereby is turned into a real promise. The recipient of the

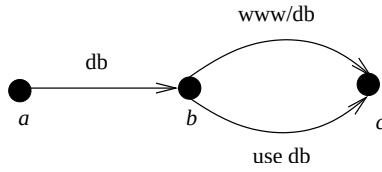


Fig. 5. Turning a conditional dependency into a real promise. A common solution is shown in graphical form.

use-promise depends somewhat on the situation. This is but one example. The question depends on who needs assurance to complete their belief in the promise.

What is important about this definition is that the logically equivalent promise is unaware of the dependency. Agent a has not promised anything to node c . This will have important implications for chains of dependencies. We state the following theorem without proof.

Theorem 3 (Forward chain theorem): Let $n_1 \xrightarrow{\pi} n_0$ be a promise that depends on a chain of agreements with a number of other nodes. The resultant promise to n_0 comes only from node n_1 and locality is preserved (see fig. 6)..

$$n_{m+1} \xrightarrow{\pi_m} n_m \bigotimes_{i=m}^{i=2} \left(n_i \xrightarrow{\pi_{i-1}/\pi_i} n_{i-1} \oplus n_i \xrightarrow{U(\pi_i)} n_{i-1} \right) \Rightarrow; n_1 \xrightarrow{\pi_0} n_0 \quad (20)$$

or

$$n_{m+1} \xrightarrow{\pi_m} n_m \bigotimes_{i=m}^{i=2} n_i \not\xrightarrow{\pi_{i-1}/\pi_i} n_{i-1} \Rightarrow n_1 \xrightarrow{\pi_0} n_0 \quad (21)$$

The proof of this result follows straightforwardly from the definitions above, by induction.

Proof: Start by relativising eqn. (18) and write for two cases:

$$\begin{aligned} n_{m+1} &\xrightarrow{\pi_m} n_m \otimes n_m \not\xrightarrow{\pi_{m-1}/\pi_m} n_{m-1} \Rightarrow n_m \xrightarrow{\pi_{m-1}} n_{m-1} \\ n_m &\xrightarrow{\pi_{m-1}} n_{m-1} \otimes n_{m-1} \not\xrightarrow{\pi_{m-2}/\pi_{m-1}} n_{m-2} \Rightarrow n_{m-1} \xrightarrow{\pi_{m-2}} n_{m-2}. \end{aligned}$$

Combining these and iterating leads to eqn. (21). ■

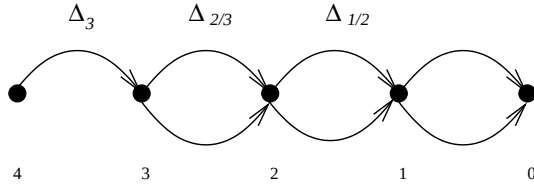


Fig. 6. Chaining of agreements. A final promise can depend on any number of chained agreements (dependencies), but these are not visible to the end node and the end node receives no direct guarantee from any of the dependencies.

D. Promise example

An example illustrates how promises force a disciplined way of thinking about systems, as a result of the constraint of autonomy. In fig. 8, we use promises to model the behaviour of the configuration management system cfengine[11], [42]. This software is designed for use in pervasive scenarios.

A particular problem for autonomous agents involves how one makes chain-like dependencies: in this case, a decision fork. The requirement of making conditional chains of promises is often used as an argument for centralization of control in systems. In cfengine, such chains of promises can be made by voluntary cooperation alone. If one promise is made and exploited, it can trigger a conditional promise (see fig. 7). In cfengine quasi-syntax, one writes:

```
active_class::
```

```
    <object-0> promise=constrain-0 define=yes
```

```
yes::
```

<object-1> promise=constrain-1

!yes::

<object-2> promise=constrain-2

One might at first expect this scenario to look like the dependency chain in fig. 7:

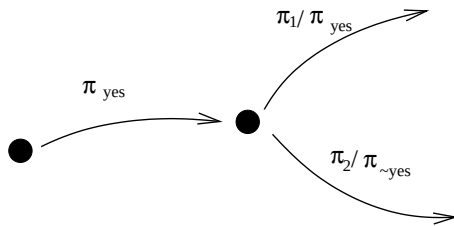


Fig. 7. We might expect the conditional structure above to look like this diagram, but we must remember that all configuration promises are between objects and cfagent.

However, this picture is not correct: promises are made between autonomous objects and an external observing agent (the cfagent). The implied chain of obligation (nodes shooting wish-arrows) in fig. 7 is not enforceable. Instead, we must have the picture in fig. 8, which shows more clearly the bilateral nature of the relationships, and indeed the necessity of having an external mediating component to broker the conditional promises.

Communication in cfengine occurs through ‘classes’. Cfengine classes are service-promises, provided by the monitoring agent to the resource monitor. It is a message-passing service. The appearance of a bilateral interaction between collaborating agents turns out to be a theme of some importance in promise theory, which has rather deep consequences for management, and we return to discuss this in ref. [38].

Finally, one can illustrate the use-promises as an example of remote file synchronization (copying) in cfengine (see fig. 9). Here a source promises to provide data, and the receiver

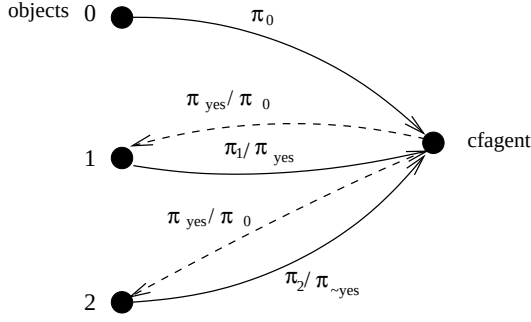


Fig. 8. The initial object makes a promise. The agent then reports back to each object if the promise is kept a values of “yes” with a broadcast promise, to each object, of the value of a variable “yes”. The two lower objects then make their own conditional promises based on whether the promise is “yes” or “!yes” (not yes).

promises to use the available source data, as well as to report its current state to the monitoring agent.

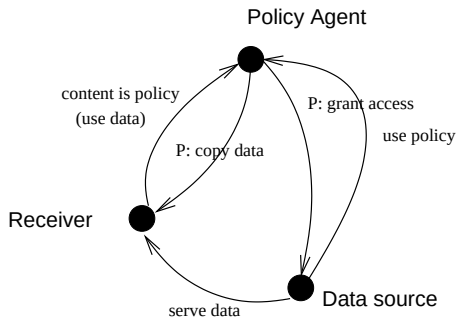


Fig. 9. Autonomous file copying in cfengine, represented as declarative promises. The copy-destination agent promises the monitor ‘cfagent’ that its file will be the same as the source agent’s file. The source agent promises to deliver the file on request. The destination agent promises to use the data from the source. At no time is anything forced onto the agents.

Let the nodes be denoted $\langle S, R, A \rangle$, for source, receiver and agent. Let P_S be S ’s policy and P_R be R ’s policy.

$$\begin{aligned}
 & A \xrightarrow{P_S} S \otimes A \xrightarrow{P_R} R \otimes \\
 & S \xrightarrow{U(P_S)} A \otimes R \xrightarrow{U(P_R)} A \otimes P \xrightarrow{U(\text{data})} A \otimes S \xrightarrow{\text{data}} R
 \end{aligned} \tag{22}$$

In each case, the nodes maintain their autonomy. There is no need for any ‘push’ or override

semantics, i.e. no need for external authority to achieve file synchronization.

IV. REGIONAL POLICIES: GROUPS AND ROLES

In this section we show how promises lead to a natural topological understanding of policy in a network. Networks cooperate and rationalize their activities by delegation and role assignment. We need to understand how such concepts arise from autonomous networks.

Roles have been defined many times in computer science. Bishop defines them to be groups[43] that tie membership to function. When a subject assumes a role, he or she is given certain skills or rights within the system. This delegation of rights and capabilities is performed by some central monitor, usually a system kernel. A role, according to Canright and Engø[44] on the other hand, is a type of node that defines the topology of the total graph. A role can be source, sink, bridge, a member or a region, or a dangler (a dead end that is both a source and a sink).

We consider roles to be associated with *types* and categories of promises π . In this respect, we can associate a group with a role, given that the members of the group make (or receive) the same type of promise to (from) some third party, or set of other parties. In other words, we are guided by the typed structure of the graph alone.

For example, web servers, clients, etc are roles. Server, client, etc are other types of roles. Roles are effective labels for groups of nodes. In this respect, we can define a group as follows.

We can define notation for a promise from one set of nodes to another set of nodes by:

$$\{n_i\} \xrightarrow{\pi} \{n_j\} \quad (23)$$

This, however, does not really make sufficient sense for nodes that are autonomous, because we do not know, in advance, whether the ‘promiser’-nodes will keep the same promise to the ‘promisees’, unless they have agreed to do so (see proposition (1)).

Roles arise in several ways. In topological terms they are typed generalizations of the concept of sources or sinks in a graph.

- An *appointed role* arises when a node or group is being pointed to by a set of identical promise arrows (role-sink). In a sense, the nodes vote on who has a particular role (like an election), by promising their allegiance π .
- Roles by *association* are uncoordinated agents that happen to make the same promises.

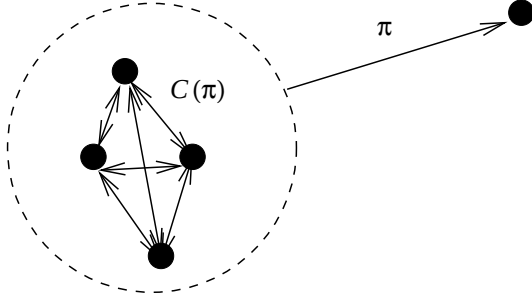


Fig. 10. A cooperative role is defined when a group, formed through mutual cooperation, agrees to offer the same promise to another node, or set of nodes. The nodes that are pointed to have a role that is appointed by the fact that they are all promised the same thing from another group of nodes.

- A *cooperative role* arises when a group of nodes is united by making a coordination/subordination promise to one or more other nodes. They take on the role of a ‘community’ or ‘organization’ of dependables.

Definition 8 (Role): Let A be a set of autonomous agents. The family of sender/receiver subsets $S, R \subseteq A$ of in a promise graph, which take part in promises with bodies Π , define Π -roles. Any such subset S or R in the graph is said to play a role of (sender/receiver) type Π in the promise graph.

Example 4: The set of all agent nodes $W \subseteq A$ that promises to provide web service to any agent:

$$W \xrightarrow{\text{web}} \quad (24)$$

plays a role, which we can call ‘web server’. This is called a role by association, since the members of W are not connected in any other way than they make the same promise.

Definition 9 (Coordinated Group): A subset of nodes $\mathcal{G}_c \subset \mathcal{N}$ is defined to be a cooperative group, if and only if each node, in $\mathcal{G}_c$, participates in a mutual agreement to coordinate with every other node in the promise:

$$\mathcal{G}_c \xrightarrow{\pi} X \quad (25)$$

for some node or set of nodes X . Every node in the group has a promise $\mathcal{G}_c \xrightarrow{C(\pi)} \mathcal{G}_c$, for the given π . (See fig. 11)

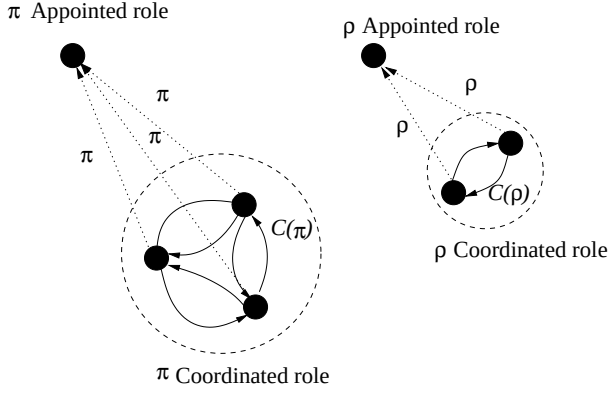


Fig. 11. Typed roles distinguish between the type of promise used to define the structure. Natural groupings result from the autonomous behaviour.

Definition 10 (Coordinated role R_c): A coordinated group, by implication, must also promise one or more external nodes π . The label-pair $R_c(\mathcal{G}_c) = (\pi)$ may be used to label the *cooperative role* of the group $\mathcal{G}_c$. (see fig. 10)

i.e. nodes are unified into a group by virtue of their making the same promise to an external node, or set of nodes. Only a complete sub-graph of collaborative agreements can make a collaborative group.

Example 5 (Load balancing): An example of what it means to build a collaborative structure is shown in fig. 12. This picture may be interpreted in a number of ways. Let us suppose that each node is a PC, and that this represents a typical search-engine type configuration built from these off-the-shelf PCs. To give the client a consistent service, the PCs in the load balancer have to agree to harmonize their policies with respect to the client (remember that they are autonomous and that this is, by no means, automatic). The four database servers are also autonomous and have their own policies (perhaps because they belong to different departments). They have not agreed to harmonize their policies. Even though they play a specific role in the network, they do not form a group, because there is nothing to bind them. If the load balancing collective agreed to “use” all of these servers, that would make them an appointed group.

V. EXAMPLES

In ref. [45], we have provided a detailed example of the promises that might be made in a small, pervasively equipped shopping mall. The promises and types are illustrated in fig. 13 as

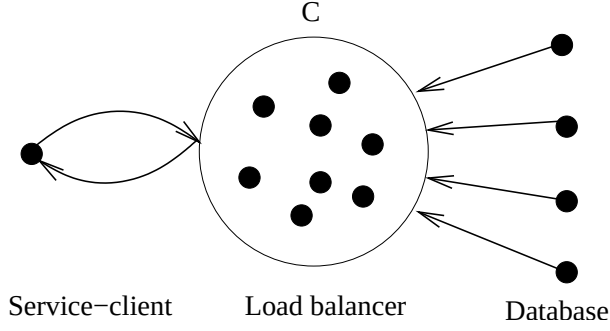


Fig. 12. A promise graph of a typical load balancing configuration. The circular region represents a fully connected graph of collaborative agreements, making all the indexing agents behave equivalently towards a service-client. Their mutual agreements define their role in this collaboration. These in turn are promised individually service by four database servers. Notice that these do not have bi-directional promises, so they might not even be used in the current policy. This might be considered an error of configuration.

different marked lines. We do not have the space here to discuss this model in detail, however, we note that this mesh of promises can be analysed straightforwardly, with this theory, to unravel the relationships and weaknesses within it. Readers are referred to ref. [45] for a detailed discussion.

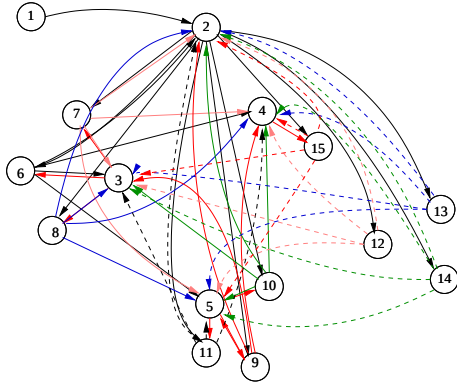


Fig. 13. A example graph of typed promises between fifteen agents.

Applying the role definitions, of section VII, to the tangled web in fig. 13, results in the simplified, functional view in fig. 14. This diagram exhibits the various relationships between the agents in operational terms. Continuing, by calculating a common currency graph for node reliability, based on held agreements, leads to a ranking of the reliability of the agents. The

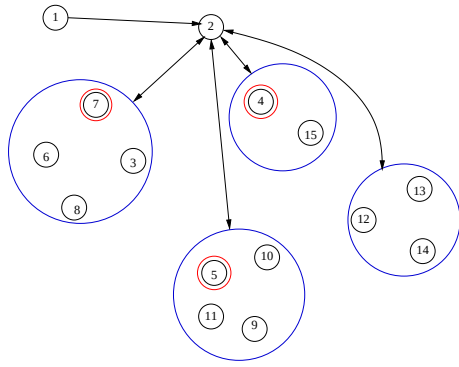


Fig. 14. A simplified schematic graph based on the derived roles from the previous figure.

ranking reveals that the system is particularly vulnerable to the failure of agent 2. This weakness is due to its central place in the graph. However, what is interesting is that, even if we delete this node (indicating perhaps its failure to fulfill a role within the system), the theory shows that the remaining promise graph survives as a functioning entity.

Fig. 15 shows that the removal of this node leaves a connected region of the system, with functioning roles, and a single isolated node, which represents the ‘outside world service provider’ in the model of ref. [45].

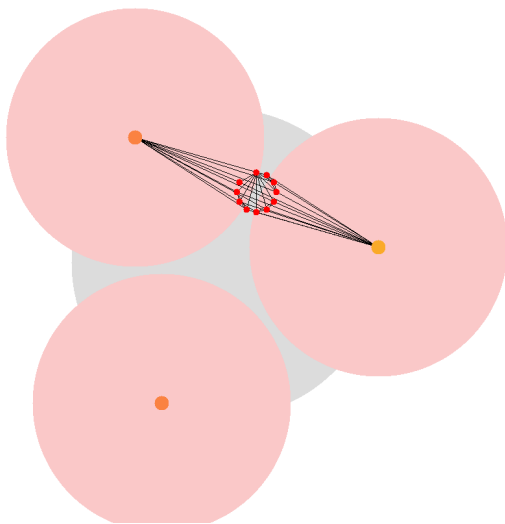


Fig. 15. Archipelago view of the network. Eliminating the mall node 2 isolates the ISP node and forms two network regions dominated by two of the shops and bridged by the intermediary customers.

VI. A THEORY OF MANAGEMENT

We close this brief overview of promises with an important corollary: how does our model of policy assist in managing a pervasive environment? We believe that management is about the ability to guide systems in approximate terms, using incentives to entice autonomous change, without the necessity complete information about system specification and state. Unless the amount of information required to formulate a policy is considerably less than the total information describing the state of the system, one must surely consider the system to be unmanageable[46]. We would therefore like to examine ways of coarse graining, or eliminating detail from the information about policy in a way that is faithful to the approximate estimations of sustainability, but which requires far less detail.

A policy, based on promises, leads to a labelled graph over possibly disjoint or partially connected sets of agents in different regions. Can one take such graph and replace it with a graph that ‘captures the essence’ of this picture? The economics of promises, is type-less, as we have already discovered. To ‘manage’ this scenario we need to be able to answer questions like:

- Are there any logical errors or missing agreements in the network?
- Which nodes are providing services with acceptable SLAs?
- What roles are necessary to distinguish or extend the functionality?
- Where are the vulnerable parts of the cooperative network?
- How much incentive do I need to provide to guarantee cooperation from an agent?
- What is the probability that a failure will be fatal?

Our theory addresses these questions directly. The first few points should already be clear, within the scope of this paper. We turn next to valuations of the promises that can tell us about the stability of the network as a functional entity.

A. *The Common Currency Promise Graph and Probable Compliance with Policy*

Each promise is associated with a topic or a theme and a constraint, the details of which have been largely suppressed in this paper for clarity. Thus each edge or link in the promise graph is labelled. Promises with different labels are not directly comparable, but it is useful to see the extent of larger classes of promise within the graph, in order to model the collective behaviour of the autonomous agents (if the agents are not really autonomous, then this gives a convenient management perspective).

For example, there are promises of the form “I offer you a web service”, “I will store your data”. We can think of the basic promise types as *service types*. There is a number of these distinct types that are, in principle, separate and distinct. and call these S . We have also identified promises of collaborative types denoted C and usage types denoted U . These can be joined together into a common graph by a mapping which conceals the labels, e.g. using a function that maps promises or types into the reals. Thus, we can construct a weight for each promise in an unlabelled graph, in a variety of ways.

This implies that there is a common ‘currency’ in any connected policy graph[46], which evaluates how much a promise is ‘worth’ to our management model. All variables can and must be expressed in terms of a common scale of measurement, i.e. there exists an equivalent graph of directed relationships, at each moment, that has no labels, only weighted edges.

Definition 11 (Common Currency Equivalent Graph): A common currency equivalent graph (CCEG) is a snapshot of the labelled promise graph, in which each promise is reduced to a single dimensionless value which can be summed.

The value of the currency can be defined as any linear combination of functions which take a label and map it into the reals. Then the common currency graph has an adjacency matrix of the general form:

$$\begin{aligned} A_{ij} &= \alpha_1 f_1(n_i \xrightarrow{\pi_1} n_j) + \alpha_2 f_2(n_i \xrightarrow{\pi_2} n_j) + \dots \\ &= \alpha_1 f_1(A_{ij}^1) + \alpha_2 f_2(A_{ij}^2) + \dots \end{aligned} \quad (26)$$

where the promises range over whatever subclass or subset of promises we wish to consider. An obvious candidate for reliability studies is to make the common currency of the graph “probability that a promise is kept”. In this case, the graph has the features of an epidemic process[47], in which the function along each link is the probability of infection given that the original node is infected. One can then see how efficiently policy ‘infects’ the agents.

B. Importance metric for resource management

Any non-labelled graph can be represented by a matrix A_{ij} called the adjacency matrix, in which the indices i, j of the matrix run over the node identities in the graph. A non-zero element $A_{ij} \neq 0$ means that there is an edge or link in the graph from $i \rightarrow j$. The value of the element is a weight associated with that edge. Often these weights are simply 1 or 0.

A labelled graph can, of course, be represented as a matrix of labels, but this limits the ability to use the matrix for numerical reasoning. An undirected graph has a symmetrical adjacency matrix.

One can formulate a precise definition of how constrained autonomous agents are by their neighbours, using the topology of the graph as a guide. In particular, the concept known as *eigenvector centrality* can be used to discover useful perspectives for management by ranking the nodes in a graph according to a specific interpretation of the links[48], [49], [46].

If one multiplies a row of the adjacency matrix, of ones and zeroes, by a column vector of ones, this simply counts the number of promises which that node exchanges with its neighbours. One can use this fact to self-consistently sum the promises from the entire graph and see which agents are most entangled. By rescaling, this leads to a unique ranking in terms of the connectivity.

Since the labelled edges of the promises lead to many typed graphs, one must decide how to deal with the different labels: whether to separate them or remove them (see next section).

Let v_i denote a vector for the cooperative ranking of each node i . Then, the importance of node i is proportional to the sum of the importances of all of i 's nearest neighbours $\mathcal{N}(i)$:

$$v_i \propto \sum_{j \in \mathcal{N}(i)} v_j . \quad (27)$$

Writing this more compactly gives

$$v_i \propto \sum_j A_{ij} v_j , \quad (28)$$

where A is the adjacency matrix, and hence:

$$A \vec{v} = \lambda \vec{v} . \quad (29)$$

The ranking vector is seen to be an eigenvector of the adjacency matrix A . If A is an $N \times N$ matrix, it has N eigenvectors (one for each agent in the network), and correspondingly many eigenvalues. The eigenvalue of interest is the principal eigenvector, i.e. that with highest eigenvalue, since this is the only one that results from summing all of the possible pathways with a positive sign. The components of the principal eigenvector rank how self-consistently ‘central’ a node is in the graph. Here, it signifies how many promises an agent is involved in. The highest valued component is the most “central”, i.e. the most entangled in promises.

C. Applications

Applications of the centrality have been given in refs. [50], [44], [46] and further applications for directed graphs are studied in ref. [40]. We note some further points:

- First of all this decomposition shows us which are the connected parts of the graph, i.e. which agents are inter-related at all?
- In a symmetrical, undirected graph A and A^T have the same eigenvalues and eigenvectors. However, if there are directed links (corresponding to unresolved agreements etc.) then the ranking is different.
- Sources and sinks, signifying incomplete agreements, show up readily in these graphs and are thus easily identifiable with a simple method.
- A^T gives a higher ranking to the recipient of a promise, whereas A gives a high ranking to the provider of a promise.
- If one takes the undirected common currency graph of all collaborative “ $C(\pi)$ ” promises, this shows us the collaborations, social structures or roles amongst the agents. Tightly bound regions will form topological roles[44] and the highest ranking node will be that which is most important in binding the collaborative structure. This is both an agent that one should be wary of and which should have enough resources to fulfill its contractual obligations.
- If one takes the common currency graph of all promises of type π and $U(\pi)$ together, these represent dependency relationships. Even though the promises only have meaning between pairs of nodes, the viewpoint offered by the eigenvector is the level of entanglement of the agents in a collaborative network. Again, this allows one to focus attention on agents where problems more likely will arise. One now has a choice between ranking by A^T or by A .

VII. EXAMPLE: SOFTWARE MANAGEMENT

Let us draw another parallel with autonomous pervasive computing management, namely software engineering management, and show how promise theory recovers some well-known insights into the structure of programming objects[51]. (This example is subtly different to a ‘real’ network of autonomous devices, because program flow is driven forward by an external agent (the system scheduler), so the autonomy of the objects is at a semantic level. Objects keep their promises without a mutually economic incentive to do so.)

A service view of software has been developed in detail, especially since the example of object orientation, in particular Smalltalk. Software is a collection of objects, including storage and methods, which perform services for one another like ‘permission to access’ or ‘transfer data’. The theory of object orientation has been built up very much around the tree-like hierarchy, but developments such as Aspect Oriented Programming underlines the fact that computer programs are not well modelled by strict hierarchies.

Our theory of promises can be applied to programming by treating every addressable item in a program as a node in a graph. By explicitly adding arrows for the services that bind the objects together, one obtains a graph of the necessary and sufficient relationships between the objects. The traditional approach to deciding encapsulation is one of doctrine, but our approach yields an answer to the natural boundaries based only on the roles in the promise graph. This graph is constructed only with the principle that *all* objects are private (autonomous) unless they promise to give up that autonomy in order to cooperate with another. We provide only the smallest example here as a taster of this approach, to be expounded elsewhere.

How do we form such a graph? Consider a simple programming assignment: $a = b$. This is a graph with two nodes a and b and two promises:

$$\begin{array}{ccc} b & \xrightarrow{\text{value}} & a \\ a & \xrightarrow{\text{Use}(\text{value})} & b \end{array} \quad (30)$$

i.e. a is promising to yield its value to a , and a is promising to use it. Function calls are services that promise to *use* data passed as arguments, and promise to *return* values as a service in an assignment. By drawing all of these functional entities as nodes (at the appropriate level of abstraction for the problem one is trying to solve), and then by filling in the necessary promises, the structure emerges without preconception.

Consider a well-used example of the ‘logging problem’. This is often used to motivate aspects and global objects such as *singleton design patterns*. Consider a program that has to log at different levels of importance (e.g. normal, verbose, debug, etc.) to a variety of output streams (e.g. console output, the system log service, files etc.). Output belonging to the debugging level should only be forwarded if system debugging is switched on. There must therefore be some state information that promises to inform of the output level that has been selected. Similarly, the program specifies the importance of each message.

```
output(debug, 'Debug only message');
```

This means the output service must promise to use both the declared level of the message and the value of the user options. It must then promise to deliver the appropriate data to the correct collection of basic streams. Notice that Use-promises are equivalent to access controls and are therefore linked to encapsulation mechanisms in languages and operating systems. A solution to this issue is shown in fig. 16, with comments. Given this graph, we can apply the common

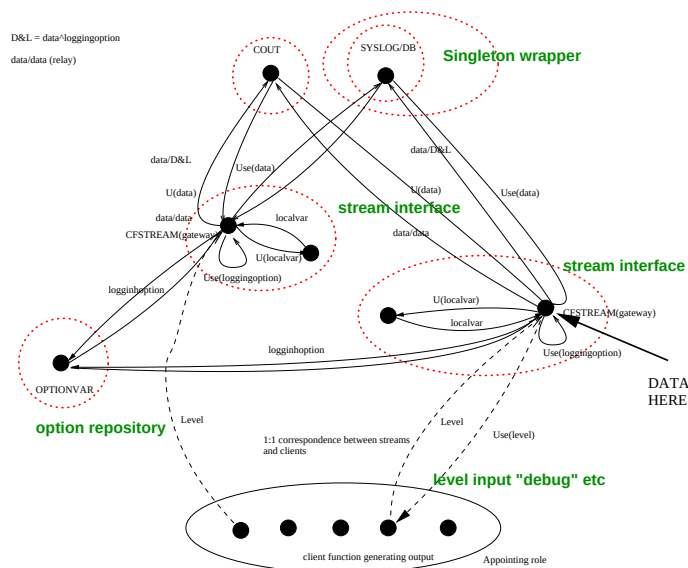


Fig. 16. Promise graph of the interaction between an adaptive output stream and its underlying resources. The ovals show the derived roles and regions, indicating the minimal form of encapsulation that follows from the labelled promises. We show two stream instances in the centre of the diagram, but this will grow to many for each different context of a program.

currency metrics in a simple way. Even from this small graph, the eigenvector centrality measure makes the final destinations (COUT/SYSLOG) into the ‘most important’ nodes. This means that they will be the most depended on. If one increases the amount of input specialization data sources, the stream interface nodes grow in importance, showing how the centrality metric gives an impartial view of the amount of derived workflow through the different objects. The reliability of the program will clearly depend on the reliability of the nodes and links. The importance metric shows us where to concentrate most of the attention on robustness[52].

Such a software graph can clearly become very complicated, perhaps prohibitively so, if one tries to bite off too much without special tools. But even in this single issue it provides a neat

solution based on a straightforward recipe.

The management of program design involves making a large number of organizational decisions about objects and their levels of privacy and encapsulation. Promise theory offers a method for addressing this issue based on minimalism. We call the result a *policy* for design. One sees that computer programs are examples of pervasive services. This idea must be explored further elsewhere.

VIII. CONCLUSIONS AND DISCUSSION

Clearly the notion of building a picture of network policy by combining multitudinous autonomous promises is both complex and overwhelming in its detail. Does this mean that this approach is impractical or worthless? We believe not.

As a management tool for understanding and designing policy, it has attractive properties; it makes hidden assumptions explicit and thus it helps one to unravel the requirements for solving different problems. Promises assist one in assessing the uncertainties remaining, since they take a more realistic view of cooperation within a network than many ‘software driver’ models.

There is a close similarity between the promise approach and the operator approach used by the autonomous agent cfengine, in configuration management. Although cfengine is not explicitly concerned with network management, its *modus operandi* is to build a policy from atomic promises about certain resources available on computers that are optionally linked by a network[11].

Cfengine builds up consistent and stable structures by imposing predictable properties through the application of independent, atomic operations, satisfying certain constraints[53]. These rules can be considered at promises between the cfengine agent and the resources. By making the building blocks primitive and having certain properties, one can prove certain types of consistency.

If policy is really about *autonomous* statements of intent, why are we bothering to show how to put them together to build larger structures? It is only by studying such effects that one can find out which policies are robust to damage. Ideally, one might be able to determine what a “typical node” should make as its “typical promises”.

By translating the detailed graph into a graph of common currency, we can make certain approximations and predictions about their meaning. For example, research on eigenvectors of

directed graphs suggests that one might be able to reduce some complex questions to very simple tests[40]. We hypothesize the following:

Hypothesis 1 (CCEG minimum requirement): The minimum requirement for a network policy to be sustainable across all affected nodes is the existence of a positive definite principal eigenvector for the adjacency matrix of the common currency equivalent graph.

Further work is required to answer this question. However, if true it would be remarkable, because there is a second motivation for the common currency graph, namely the economic or cooperative motivation[38]. Our preliminary studies do not contradict this hypothesis.

We are now in a position to be able to attempt to answer vital questions in future work. How stable is the network to cooperation? Could it ever be completely stable or completely unstable? The models of voluntary cooperation indicate that policy should be formulated in a way that builds both rewards and punishments into interactions between peers in order to secure stability.

In this paper we are afforded just enough space to lay the foundations for studying the uncertainties inherent in pervasive computing ‘management’. It remains to be seen how easily these concepts can be harnessed to develop stable rule-based cooperatives in the real world. We return to these details in the subsequent papers.

This work is supported by the EC IST-EMANICS Network of Excellence (#26854)

REFERENCES

- [1] K. Henricksen, J. Indulska, and A. Rakotonirainy, “Modeling context information in pervasive computer systems,” in *Lecture Notes in Computer Science*, 2002, vol. 2414, pp. 167–180.
- [2] L. Kagal, T. Finin, and Joshi Anupam, “A policy language for a pervasive computing environment,” in *IEEE 4th International Workshop on Policies for Distributed Systems and Networks, 2003. Proceedings. POLICY 2003*, 2003, pp. 63–74.
- [3] I W Marshall and L Sacks, “Autonomous Management for Pervasive Computing,” in *Proceedings ICCS*, Nashua, unknown 2002.
- [4] Naftaly Minsky, “Law governed interaction (lgi): A distributed coordination and control mechanism (an introduction, and a reference manual),” Tech. Rep., Rutgers University, June 2005.
- [5] N. Minsky and P-P. Pal, “Law-governed interaction: A coordination and control mechanism for heterogeneous distributed systems,” *ACM Transactions on Software Engineering and Methodology*, vol. 9, pp. 273–305, 2000.
- [6] F. Mattern and M. Naghshineh, Eds., *Pervasive Computing*, vol. 2414 of *Lecture Notes in Computer Science*. Springer, 2002.
- [7] R. Axelrod, *The Evolution of Co-operation*, Penguin Books, 1990 (1984).
- [8] F.R.K. Chung, “Spectral graph theory,” *Regional Conference Series in Mathematics, American Mathematical Society*, vol. 92, pp. 1–212, 1997.

- [9] M. Sloman, "Policy driven management for distributed systems," *Journal of Network and Systems Management*, vol. 2, pp. 333, 1994.
- [10] M.S. Sloman and J. Moffet, "Policy hierarchies for distributed systems management," *Journal of Network and System Management*, vol. 11, no. 9, pp. 1404, 1993.
- [11] M. Burgess, "A site configuration engine," *Computing systems (MIT Press: Cambridge MA)*, vol. 8, pp. 309, 1995.
- [12] E.C. Lupu and M. Sloman, "Towards a role based framework for distributed systems management," *Journal of Network and Systems Management*, vol. 5, 1996.
- [13] 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*, p. 293, 2003.
- [14] M. Sloman and K. Twidle, *Network and Distributed Systems Management*, chapter 16, Domains: A Framework for Structuring Management Policy, Addison Wesley, Harlow, 1994.
- [15] J.D. Moffett, *Network and Distributed Systems Management*, chapter 17, Specification of Management Policies and Discretionary Access Control, Addison Wesley, Harlow, 1994.
- [16] E. Lupu and M. Sloman, "Conflict analysis for management policies," in *Proceedings of the Vth International Symposium on Integrated Network Management IM'97*, May 1997, pp. 1–14, Chapman & Hall.
- [17] R. Ortalo, "A flexible method for information system security policy specifications," *Lecture Notes on Computer Science*, vol. 1485, pp. 67–85, 1998.
- [18] J. Glasgow, G. MacEwan, and P. Panagaden, "A logic for reasoning about security," *ACM Transactions on Computer Systems*, vol. 10, pp. 226–264, 1992.
- [19] H. Prakken and M. Sergot, "Dyadic deontic logic and contrary-to-duty obligations," in *Defeasible Deontic logic: Essays in Nonmonotonic Normative Reasoning*, vol. 263 of *Synthese library*. Kluwer Academic Publisher, 1997.
- [20] A.K. Bandara, E.C. Lupu, J. Moffett, and A. Russo, "A goal-based approach to policy refinement," in *Proceedings of the 5th IEEE Workshop on Policies for Distributed Systems and Networks*, 2004.
- [21] A.K. Bandara, E.C. Lupu, J. Moffett, and A. Russo, "Using event calculus to formalise policy specification and analysis," in *Proceedings of the 4th IEEE Workshop on Policies for Distributed Systems and Networks*, 2003.
- [22] A.L. Lafuente and U. Montanari, "Quantitative mu-calculus and ctl defined over constraint semirings," *Electronic Notes on Theoretical Computing Systems QAPL*, pp. 1–30, 2005.
- [23] J. Bergstra, A. Ponse, and S.M. Smolka, Eds., *The Handbook of Process Algebra*, Elsevier, 2001.
- [24] J. Parrow, *An Introduction to the π -Calculus*, in *The Handbook of Process Algebra*, p. 479, Elsevier, Amsterdam, 2001.
- [25] R. David and H. Alla, "Petri nets for modelling of dynamic systems — a survey," *Automatica*, vol. 30, pp. 175–202, 1994.
- [26] B. Jonsson, W. Yi, and K.G. Larsen, *Probabilistic Extensions of Process Algebras*, in *The Handbook of Process Algebra*, p. 685, Elsevier, Amsterdam, 2001.
- [27] Mark Burgess, "An approach to understanding policy based on autonomy and voluntary cooperation," in *IFIP/IEEE 16th international workshop on distributed systems operations and management (DSOM)*, in *LNCS 3775*, 2005, pp. 97–108.
- [28] D. Daly, G. Kar, and W. Sanders, "Modeling of service-level agreements for composed services," *IFIP/IEEE 13th International Workshop on Distributed Systems: Operations and Management (DSOM 2002)*, p. 4, 2002.
- [29] M. Burgess, "On the theory of system administration," *Science of Computer Programming*, vol. 49, pp. 1, 2003.
- [30] C. Hewitt, P. Bishop, and R. Steiger, "A universal modular actor formalism for artificial intelligence," in *Proc. of the 3rd IJCAI*, Stanford, MA, 1973, pp. 235–245.

- [31] M. Wooldridge, *An Introduction to MultiAgent Systems*, Wiley, Chichester, 2002.
- [32] J. Ferber, *Multi-agent Systems: Introduction to Distributed Artificial Intelligence*, Addison Wesley, 1999.
- [33] A. Høyland and M. Rausand, *System Reliability Theory: Models and Statistical Methods*, J. Wiley & Sons, New York, 1994.
- [34] Z. Fu and S.F. Wu, “Automatic generation of ipsec/vpn security policies in an intra-domain environment,” *Proceedings of the 12th international workshop on Distributed System Operation and Management (IFIP/IEEE)*., vol. INRIA Press, pp. 279, 2001.
- [35] R. Sailer, A. Acharya, M. Beigi, R. Jennings, and D. Verma, “Ipsecvalidate - a tool to validate ipsec configurations,” *Proceedings of the Fifteenth Systems Administration Conference (LISA XV) (USENIX Association: Berkeley, CA)*, p. 19, 2001.
- [36] L. Snyder, “Formal models of capability-based protection systems,” *IEEE Transactions on Computers*, vol. 30, pp. 172, 1981.
- [37] R. Axelrod, *The Complexity of Cooperation: Agent-based Models of Competition and Collaboration*, Princeton Studies in Complexity, Princeton, 1997.
- [38] M. Burgess and S. Fagernes, “Pervasive computing management: Policy through voluntary cooperation,” (*preprint*), p. (submitted).
- [39] M. Burgess and S. Fagernes, “Pervasive computing management: A model of network policy with local autonomy,” *IEEE Transactions on Software Engineering*, p. (submitted).
- [40] M. Burgess, G. Canright, and K. Engø, “Importance-ranking functions from the eigenvectors of directed graphs,” *Journal of the ACM (Submitted)*, 2004.
- [41] J.A. Bergstra and A. Ponse, “Interface groups for analytical execution architecture,” (*preprint*), 2006.
- [42] M. Burgess, “Configurable immunity for evolving human-computer systems,” *Science of Computer Programming*, vol. 51, pp. 197, 2004.
- [43] M. Bishop, *Computer Security: Art and Science*, Addison Wesley, New York, 2002.
- [44] G. Canright and K. Engø-Monsen, “A natural definition of clusters and roles in undirected graphs,” *Science of Computer Programming*, vol. 53, pp. 195, 2004.
- [45] M. Burgess and S. Fagernes, “Pervasive computing management: Applied promise theory,” (*preprint*), p. (submitted).
- [46] M. Burgess, *Analytical Network and System Administration — Managing Human-Computer Systems*, J. Wiley & Sons, Chichester, 2004.
- [47] D.J. Watts, *Small Worlds*, (Princeton University Press, Princeton), 1999.
- [48] P. Bonacich, “Power and centrality: a family of measures,” *American Journal of Sociology*, vol. 92, pp. 1170–1182, 1987.
- [49] P. Bonacich, “Factoring and weighting approaches to status scores and clique identification,” *Journal of Mathematical Sociology*, vol. 2, pp. 113–120, 1972.
- [50] Tuva Hassel Stang, Fahimeh Pourbayat, Mark Burgess, Geoffrey Canright, Kenth Engø, and Åsmund Weltzien, “Archipelago: A network security analysis tool,” in *Proceedings of The 17th Annual Large Installation Systems Administration Conference (LISA 2003)*, San Diego, California, USA, October 2003.
- [51] D. Aredo, M. Burgess, and S. Hagen, “Object orientation and the service oriented architecture: A model using promise theory,” in *submitted to 3rd European Workshop on Software Architecture (EWSA 2006)*.
- [52] M. Burgess, G. Canright, and K. Engø, “A graph theoretical model of computer security: from file access to social engineering,” *International Journal of Information Security*, vol. 3, pp. 70–85, 2004.

- [53] M. Burgess, "Configurable immunity model of evolving configuration management," *Science of Computer Programming*, vol. 51, pp. 197, 2004.