Knowledge Management and Promises

Mark Burgess

Oslo University College, Norway
mark@iu.hio.no

Abstract. Ontological modelling for machine inference has featured prominently in IT management research recently, but there is more immediate scope for knowledge modelling in the realm of human inference. This work discusses the relationship between ISO Standard Topic Maps and Promise Theory and shows how these two knowledge models models complement one another and offer a *semantic* approach to policy based management that can reduce organizational information complexity.

1 Introduction

Ontological modelling has been discussed at length as a way of using domain knowledge to enhance web services for IT management. Ontology models typically involve complex XML markups intended for such smart web interfaces [1–4] but do little to aid human understanding. Unfortunately, semantic inference in automatic systems is beset with difficulties: the overhead in both modelling and processing tends to lead to benefits that are either simplistic to humans or have problems that are intractable to machines. The reason for this seems to be that semantics are the domain of human imagination, not of machine logic. This suggests a return to a division of labour between humans and machines in which each party does what is most natural to it. Humans are good at reasoning and associative thinking when provided with quality information, while machines are good at consistent, repetitive implementation and rather poor at reasoning. This is the starting point for this work.

Semantic modelling of information pre-dates the Web by several years in fact. Topic Maps, discussed in this work, were originally invented as a form of electronic book-index, enhanced with associative thinking [5]. They have since fallen into the shadow of efforts surrounding the semantic web research, but wrongly in the opinion of the author. Topic maps appear simpler than the general ontology languages of the semantic web and they are designed for human appraisal rather than machine inference. Topic maps also have a simple relationship with the theory of promises, which this paper aims to make use of.

2 Promises and topics

The concept of promises was introduced into IT management in 2005 as a fresh approach to the problems of conflicts in policy based management [6]. The heuristics and extended motivation for the model were described later in [7]. Promises provide a model for abstracting the intention behind operations from the operations themselves, and have attractively simple algebraic and semantic properties. The core of promise theory is the *autonomy* of agents that are able to make promises, and the subsequent notion of *voluntary cooperation*, replacing the problematic notion of obligation. This leads to many theoretical and practical simplifications. The reference implementation of promise theory as a technology is Cfengine [8], which forms an integral part of the discussion below. Every statement made in the essentially declarative cfengine language can be understood as being part of a promise.

This declarative manifesto, backed up by practical guarantees of outcome reachability [9], has an important implication for automated management: the complete separation of intent from action means that what remains for the human is to model *intent*. This is a form of knowledge management, and thus the focus moves from implementation to knowledge.

Knowledge management is a field of research in its own right, and it covers a multitude of issues both human and technological. Most would agree that knowledge is composed of facts and relationships and that there is a need both for clear definitions and semantic context to interpret knowledge properly; but how do we attach *meaning* to raw *information* without ambiguity? This is an ad hoc association which follows human social conventions, and is therefore poorly suited to machine reasoning.

Knowledge has much in common with configuration: what after all is knowledge but a configuration of ideas in our minds, or on some representation medium (paper, silicon etc). It is a coded pattern, preferably one that we can agree on and share with others. Both knowledge and configuration management are about describing patterns.

Previous models of management have been based on pure data modelling and the assumption 'guaranteed' change in accordance with the data [1, 10]. What makes Topic Maps attractive compared to other more complex ontology tools is that they are intended for human reasoning (something humans are very good at), not for machine inference (which is something machines have rarely been very good at). Moreover, although they sound like very different animals, Topic Maps and Promises are in fact homomorphic and complement one another in a neat, symbiotic relationship.

The reasoning may be summarized as follows: a simple knowledge model can be used to represent a simple policy configuration model; conversely, a simple model of policy configuration can represent indeed manufacture a knowledge structure, and there is a natural promise engine that can implement this mapping: cfengine.

The Topic Map and Promise models are compatible because they appeal to the same basic world-view: principles for reduction of knowledge into atoms and the autonomy of concepts that automatically avoids overlap and conflict. Both models effectively use the idea of autonomy of entities and a simple context based data model that allows them both to represent their subjects, as well as one another, in a homomorphic way.

3 The promise model

3.1 Promise theory

Promises are a modelling framework (see [6]) that presents a decentralized view of behaviour in systems. Promise theory describes the intentions and attributes of system artefacts (i.e. anything from system components to ideas), which are autonomous in the sense that they can change independently.

A promise is the announcement of an intention [7] (usually expected to represent a possible future) and it requires verification to confirm eventual compliance. Promises are not events but conditions (states) that persist in the memory of recipients who are in the scope of the promise. A promise is more than an intention, since an intention need not be announced, and it is less than a commitment since a commitment often involves an investment or action plan for keeping the promise.

Promises are made by a promiser 'agent' to a promisee 'agent', i.e. they are directed relationships each labelled with a promise *body* which describes the substance of the promise. A promise with body $+b$ is understood to be a declaration to "give" behaviour from one agent to another (possibly in the manner of a service), while a promise with body $-b$ is a specification of what behaviour will be received, accepted or "used" by one agent from another (see table 1). A promise *valuation* $v_i\left(a_j \xrightarrow{b} a_k\right)$ is a subjective interpretation by agent a_i (in a currency of its choice) of the value of the promise in the parentheses; this can be used for ranking of importance, for example. The value can be negative if it is pure cost. Usually an agent can only evaluate promises in which it is involved.

Symbol	Interpretation
$a \xrightarrow{+b} a'$	Promise with body b
$a' \xrightarrow{-b} a$	Promise to accept b
$v_a(a \xrightarrow{b} a')$	The value of promise to a
$v_{a'}(a \xrightarrow{b} a')$	The value of promise to a'

Table 1. Summary or promise notation

A promise body b has a *type* which describes the nature or subject of the promise, and a *constraint* which explains what restricted subset of the total possible degrees of freedom are being promised. Since any dynamical, systematic behaviour is a balance between degrees of freedom (avenues for change) and constraints, this is sufficient to describe a wide variety of phenomena.

Promise theory is mainly about the analysis of epochs in which promises are essentially fixed. If basic promises change, we enter a new epoch of the system in which basic behaviours change. Thus promise theory is mainly about steady-state behaviour about which one can accumulate lasting knowledge.

3.2 Promises in cfengine

Cfengine 3 is the reference implementation of promise theory as a technology for configuration. It allows promises to be expressed as a language and kept by software automation. Cfengine's representation of a promise has the following generic form:

```
promise-type:

   context-classifiers::

      promiser-object -> { list of possible promisees },

         comment => ‘‘Expression of intention’’,
         body-attribute-1 => value-1,
         ...
         body-attribute-n => value-n;
```

Such declarations are grouped into ‘promise bundles’. Such a statement encodes a single promise, for example:

```
files:

   linux||solaris::

      "/etc/shadow" -> { "ISO17799 team", "other promise" },

         comment => "Check integrity of the shadow file",
         changes => record_hash_changes(),
         perms   => my_perms("root","0600");
```

This partially disclosed promise is directed at a team of humans and at another promise that depends on this one. It concerns the file `/etc/shadow`. The scope or context of the promise is the set of all agents that belong to the classes `linux` or `solaris`, and the body of the promise contains the properties that the file should have: namely a particular owner, a particular set of access rights to the file and a static hash signature.

Cfengine views this initial declaration as a *promise proposal*, which is intended to apply to any host or agent in scope. The declaration of the suggested promise is typically made at some central location where management is centralized.

Other hosts that are not represented in the scope are supposed to ignore the promise proposal, but hosts that lie in these classes will normally take this promise proposal at face value and try to keep it, as if it were a command, although there is no way of actually forcing them to do this. Indeed this voluntary behaviour represent another kind of retractable promise, namely one to accept these suggestions and implement them in the first place. At every step, the cooperation by agents is voluntary, but the effect is to set up an entirely prosaic workflow.

There are many different types of promise that one can make in cfengine. The engine is extensible and different agents are provided to keep different kinds of promises. The component `cf-agent`, for instance, can make promises to configure the resources of computers. With the advent of cfengine 3, a knowledge management component was introduced called `cf-know`, in which topic map relationships could be promised. Cfengine forms the basis by which one might integrate the management of declarative knowledge about configuration promises with the intentions and implementations of the declarations.

4 The topic map model

4.1 Introduction

Let us now examine the topic map model. Topic maps were originally designed for creating generalized electronic book-indices. They pre-date the World Wide Web by several years yet they work effectively as a *semantic web* of subject references and document pointers. The basic model is effectively described in terms of the TAO: Topics, Associations and Occurrences [11].

A topic is representation of any subject one wishes to discuss, abstract or physical e.g. an item of ‘abstract knowledge’, which might have a number of exemplars. It might be a person, a machine, a quality, etc. Topics may be classified into *topic-types* so that related things can be collated and unrelated things can be separated, e.g. types allow one to distinguish between `rmdir` the Unix utility and rmdir the Unix system-call. Each typed topic can further point to a number of exemplars called *occurrences*, which might include documents, database entries, physical manifestations and other information references that exemplify or are about the topic. Occurrence references are like the page numbers in an index. Unlike an ordinary index, a topic map has a rich (potentially infinite) variety of cross reference types.

A book index typically has ‘see also’ to refer from one topic to another. Topic Maps allow one to define any kind of *association* between topics. For instance,

```
 topic_1 ‘‘is a kind of’’ topic_2
 topic_1 ''is improved by'' topic 2
 topic_1 ''solves the problem of'' topic_2
```

and so on.

The topic map model thus has three levels, in order: types, topics and occurrences. These all label different levels of granularity. Types are exemplified by topics, which in turn are exemplified by occurrences (though in a different way). The primacy of topics in this hierarchy stems from their ability to form networks. Thus, while topic-type classifications and occurrences are disjoint entities, topics willfully connect in a space of associative interconnectedness.

The classic approach to information modelling is to build a hierarchical decomposition of non-overlapping objects. Entity relation models and object oriented class hierarchies force all occurrences of data to lies in a rigid information model. The data are forcibly manipulated into non-overlapping containers which often prove to be overly restrictive (cf. the need for aspect orientation and 'friends' in Object Orientation etc).

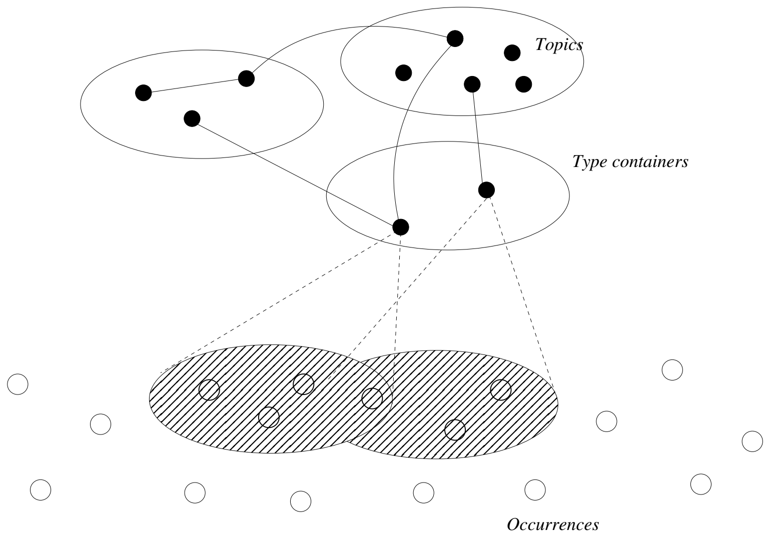


Fig. 1. Topic maps as a concept transducer: abstract topics live inside abstract classifiers or types, but can network independently. Topics 'shine different lights' on the concepts they represent called occurrences.

Each topic allows us to effectively 'shine a light' onto the occurrences of information that highlight the concepts pertinent to the topic somehow (see fig. 1). Formally topic maps use the term 'occurrence-types' to label individual topic's relationships to (viewpoints on) their occurrences; *occurrence relationship types* are not necessarily sub-types of the topics. Topics and occurrences are not classified into Draconian disjoint sets by their types: indeed, this is what allows topic maps to exceed simple hierarchical data modelling. Topic maps (networks of associated topics) bridge the world of concepts and exemplars by working as a ‘concept transducer’ that shines multiple lights onto the real world of occurrences.

The topic map model is an ISO standard that is quite rich in possibilities [5]. As a framework, the model has a simplicity to address the real problems of information complexity, but it lacks an operational road-map for usage. The same can be said about cfengine, which also provides a framework without a road-map. The key observation to integrating topic with promises is this: topic maps can represent the knowledge declared in a set of promises. They can model the relationships between the parts and types of a promise, they can point to occurrences of the promise made by multiple parties and they can discuss the abstract intentions of the promises with references to other literature. Promises on the other hand can describe how to build a topic map, its configuration and its maintenance. There is thus a natural duality between topic maps and promises.

4.2 The Cfengine Topic Map Model

Cfengine is a ‘self-healing’ management automation system, meaning that it can adapt to change and repair errors without human intervention. it was created by the author in 1993 [12]. It currently consists of a number of components that run on each individual (i.e. autonomous) computer in a network, and each computer typically voluntarily collects promise suggestions from a single point of management. The components are all able to make promises, e.g. `cf-agent` can make (and keep) promises about configuration changes, while `cf-monitord` can make promises about system data collection, etc. These components integrate to form maintenance loops.

Cfengine’s knowledge agent `cf-know` makes promises about knowledge relationships, using the model of topic maps. It is not a generic topic map language: it provides a configuration language for managing a knowledge base that can be compiled into a topic map. The full ISO standard topic map model is sufficiently rich to capture a general topic index, indeed it is almost too rich to be a useful tool for system knowledge management. However, this is where powerful configuration management can help to simplify the process: encoding a topic map is a complex problem in configuration, which is exactly what cfengine is for. Cfengine’s topic map promises have the following form:

```
topics:

  topic_type_context::                               # canonical container

    "Topic name"                                     # short topic name

            comment => "Use this for a longer description",
        association => a("forward assoc to","Other topic","backward assoc");
    "Other topic";

occurrences:

  Topic_name::                                # Topic

    "http://www.example.org/document.xyz"     # URI to instance

      represents => { "Definition", "Tutorial"}; # sub-types
```

A topic declaration involves a *type*, in topic map parlance, which maps to a scope or class context in cfengine. The topic-type is itself a reference to a container topic, since a type is also a topic. Topics of given types form disjoint sets; however topics of the same name may exist in several types, e.g. Cfengine (the software) or Cfengine (the company). Each topic effectively promises to have a name and a number of associations with other topics. In the cfengine mapping, this is made explicit. The type of a topic is simply the cfengine *class* canonicalization of the topic name used as a scope; in practice this means converting non alphanumeric characters into underscores and adding the double-colon.

The distinction between topic and topic-type is to some extent immaterial in a topic map; at one level, a type relationship is just another kind of association between topics. However, type is used differently: its describes disjoint *classifiers* (cfengine is an instance of software, or cfengine is an instance of company), where as associations are used more generally at the conceptual level to link topics of any type into a network without boundaries (cfengine 'is used to implement' promises, or promises 'may be used to configure' a topic map).

Cfengine's rendition of Topic Maps is simplified even further. It does not implement the full ISO standard, but rather a subset that is necessary and sufficient to be isomorphic with promises. Promise theory adds a clear structure to the topic map ontology, which is highly beneficial as experience shows that weak conceptual models lead to poor knowledge maps. The result is a language for making simple topic maps which (although it does not support the entire ISO standard) is both simpler and adds powerful features allowing variable expansion, re-use and more structured bundling of data.

5 Modelling configuration promises as topic maps

We can model topic maps as promises within cfengine; the question then remains as to how to use topic maps to model configurations so that cfengine users can navigate the documented promises using a web browser and be able to see all of the relationships between otherwise isolated and fragmentary rules. This will form the basis of a semantic Configuration Management Database [13] (sCMDB) for the cfengine software. The key to making these ends meet is to see the configuration of the topic map as a number f promises made in the abstract space of topics and the turning each promise into a meta-promise that models the configuration as a topic with attendant associations. Consider the following cfengine promise.

```
bundle agent update
{
files:

 any::

   ‘‘/var/cfengine/inputs’’ -> { ‘‘policy_team’’, ’’dependent’’ },

            comment => ‘‘Check policy updates from source’’,
              perms => true,
               mode => 600,
          copy_from => true,
        copy_source => /policy/masterfiles,
            compare => digest,
       depth_search => true,
              depth => inf,
           ifelapsed => 1;

}
```

This system configuration promise can be mapped by cfengine into a number of other promise proposals intended for the `cf-know` agent. Suppressing some of the details, we have:

```
type_files::

  "/var/cfengine/inputs"
      association => a("promise made in bundle","update","bundle contains promise");
  "/var/cfengine/inputs"
      association => a("specifies body type","perms","is specified in");
  "/var/cfengine/inputs"
      association => a("specifies body type","mode","is specified in");
  "/var/cfengine/inputs"
      association => a("specifies body type","copy_from","is specified in");

   # etc ...

 occurrences:

   _var_cfengine_inputs::

    "promise_output_common.html#promise__var_cfengine_inputs_update_cf_13"
       represents => { "promise definition" };
```

Note that in this mapping, the actual promise (viewed as a real world entity) is an occurrence of the topic 'promise'; at the same time each promise could be discussed as a different topic allowing meta-modelling of the entity-relation model in the real-world data. Conversely the topics themselves become configuration items or 'promisers' in the promise model. The effect is to create a navigable semantic web for traversing the policy; this documents the structure and intention of the policy using a small ontology of standard concepts and can be extended indefinitely by human domain experts (see [14]).

We end up with the following mappings, which because they are based on a model that is both simple and rigid is guaranteed to lead to densely connected networks. The two concepts may be compared as tuples:

$$\begin{aligned} \textit{Promise} &: \langle P_r, P_e^*, B \rangle, \quad B = \langle L, V \rangle^* \\ \textit{Topic map} &: \langle T, A^*, O^* \rangle, \quad A = \langle L, T_A \rangle^* \end{aligned} \tag{1}$$

For promises: P_r is the promiser, P_e^* is a number of promisees (all of which are autonomous entities). B is the body of constraints that describes the promise's intent. This in turn consists of pairs of associations between names (l-values) L of constrainable properties and the values V to be adhered to. For topic maps: T is a topic, which is an autonomous, standalone entity. A is a body of associations to other topics in topic-space, and O^* is a number of associated occurrences of the topic in the documentation/information medium. A body of associations is a set of pairs of association labels or types L and topics $T_A \in T$. There is no principled difference between associations between topics and occurrence relations, except the sets or spaces to which the end-points belong. In mapping to promises we would use associations to map concepts discussed in promises, and occurrences to map to the instances of rules in a policy.

Table 2 shows this mapping of concepts in words.

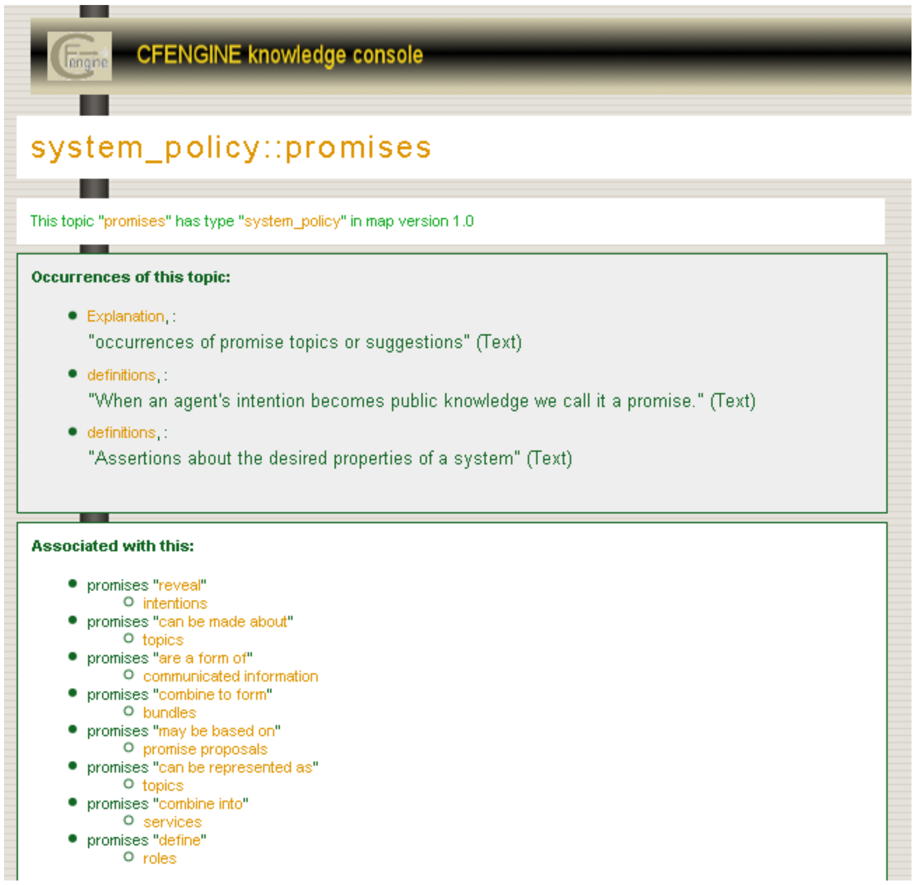
Promise theory	Topic maps
Promise of type topics	Topic
Promise of type occurrences	Occurrences
Promiser P	Topic name
Promise context	Topic-type or classifier
Promise body	Associations
Promisees	Special associations

Table 2. Mapping between promises and topic map concepts.

6 Knowledge about promises

Cfengine is able to construct the topic map and the promise graph and perform analyses of these. Figure 3 shows a simple top-level example of a navigable information structure computed by cfengine from a few topics relating to this paper (with association labels removed). Imagine the same idea as a discussion of the meaning intended in policy.

Figure 2 shows part of a page rendered by `cf-know` about the subject of promises. It should various interpretations that are promised in the configuration, and their associations to related topics. Further down this page (not



system_policy::promises

This topic "promises" has type "system_policy" in map version 1.0

Occurrences of this topic:

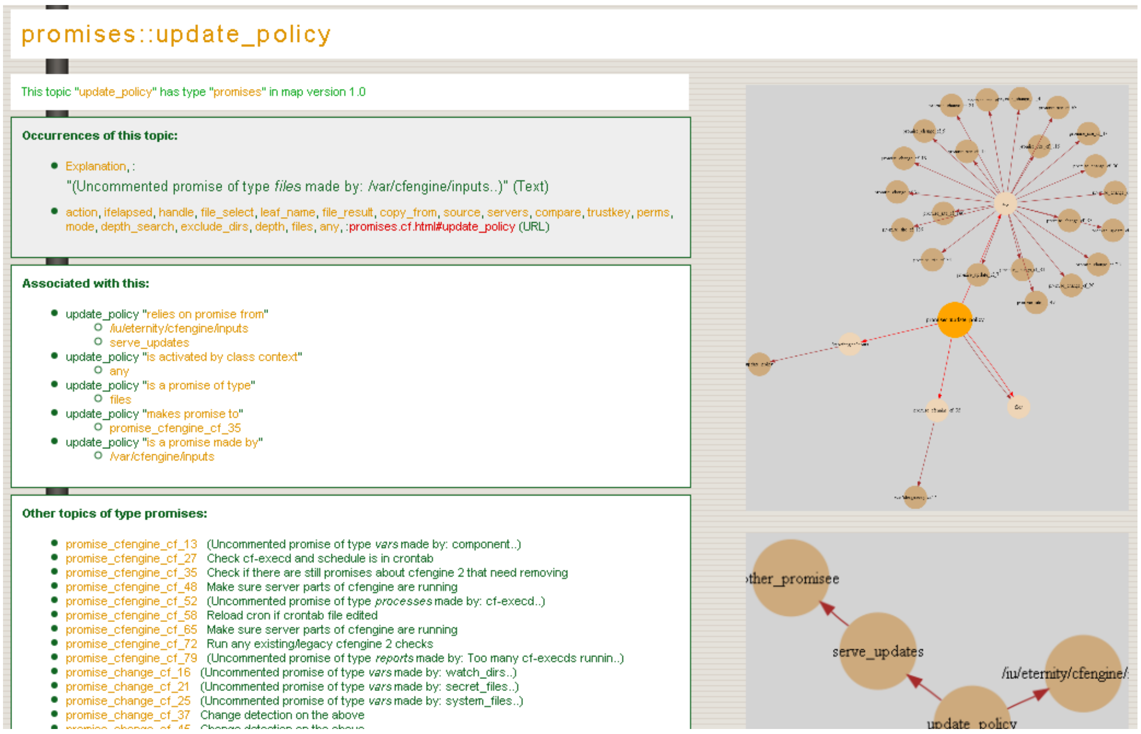
- Explanation, :
  "occurrences of promise topics or suggestions" (Text)
- definitions, :
  "When an agent's intention becomes public knowledge we call it a promise." (Text)
- definitions, :
  "Assertions about the desired properties of a system" (Text)

Associated with this:

- promises "reveal"
  - intentions
- promises "can be made about"
  - topics
- promises "are a form of"
  - communicated information
- promises "combine to form"
  - bundles
- promises "may be based on"
  - promise proposals
- promises "can be represented as"
  - topics
- promises "combine into"
  - services
- promises "define"
  - roles

Fig. 2. An excerpt of a topic page about promises, showing several occurrences of interpretation of the concept, and their associative links to related subjects.

shown) are links to topics of *type* promise. Following one of these links takes us to fig. 3, which is a configuration promise that is actually encoded as policy for `cf-agent`. The upper graph shows the thirty or so most closely associated topics (by any association), and the lower (truncated) graph picks out only *promisee*, and `depends_on` constraints, thus automatically generating a possible impact analysis for changes to any of these promises.



promises::update_policy

This topic "update_policy" has type "promises" in map version 1.0

Occurrences of this topic:

- Explanation, :
  "(Uncommented promise of type *files* made by: /var/cfengine/inputs..)" (Text)
- action, ifelapsed, handle, file_select, leaf_name, file_result, copy_from, source, servers, compare, trustkey, perms, mode, depth_search, exclude_dirs, depth, files, any, :promises.cf.html#update_policy (URL)

Associated with this:

- update_policy "relies on promise from"
  - /iu/eternity/cfengine/inputs
  - serve_updates
- update_policy "is activated by class context"
  - any
- update_policy "is a promise of type"
  - files
- update_policy "makes promise to"
  - promise_cfengine_cf_35
- update_policy "is a promise made by"
  - /var/cfengine/inputs

Other topics of type promises:

- promise_cfengine_cf_13 (Uncommented promise of type *vars* made by: component..)
- promise_cfengine_cf_27 Check cf-execd and schedule is in crontab
- promise_cfengine_cf_35 Check if there are still promises about cfengine 2 that need removing
- promise_cfengine_cf_48 Make sure server parts of cfengine are running
- promise_cfengine_cf_52 (Uncommented promise of type *processes* made by: cf-execd..)
- promise_cfengine_cf_58 Reload cron if crontab file edited
- promise_cfengine_cf_65 Make sure server parts of cfengine are running
- promise_cfengine_cf_72 Run any existing/legacy cfengine 2 checks
- promise_cfengine_cf_79 (Uncommented promise of type *reports* made by: Too many cf-execds runnin..)
- promise_change_cf_16 (Uncommented promise of type *vars* made by: watch_dirs..)
- promise_change_cf_21 (Uncommented promise of type *vars* made by: secret_files..)
- promise_change_cf_25 (Uncommented promise of type *vars* made by: system_files..)
- promise_change_cf_37 Change detection on the above

Fig. 3. A topic page about a policy item rendered by `cf-know`, showing relationships between neighbouring concepts and the subset of these that gives dependency or change-impact analysis.

Graphical representations of information open up all kinds of analyses and allow imaginative humans to see possibilities in a way that only confuses machine reasoning systems. One challenge in knowledge management is that of reducing information complexity. Even in a policy specification, the rule sets (promises) can run to thousands of lines and might contain dependencies and relationships that are not obvious to the reader. With the Topic-Map-Promise alliance, one can automatically generate a topic map from a promise-based policy and then annotate it and link it to other information bases, effectively forming a set of adaptive container classes. The chief advantage of topic maps is that they are non-hierarchical and their categories are adaptive and context dependent in the links between topic and occurrence – this is the same as cfengine's promise model and reduces the depth of information structures. Moreover there will be no categorization conflicts in either promises or topic maps by design. This sets them apart from obligation systems and Object Oriented data models.

Nevertheless, any information model that has typed elements is two dimensional and must trade complexity in the number of types and sub-types (depth) against the number of topics in each type category (breadth) and this is a fundamental problem of all information systems. However, there is a way out: the free networking of associations in topic maps means one can form standard one-dimensional pathways (routing solutions) through the associative network, which we might call 'stories'. In ref. [15] we develop this idea to find minimum distance connections between topics that are modelled for human consumption. Thus the notion of topic maps extends to include 'typed stories' based on the idea of transitivity rules for semantic associations. The homomorphism with promises further implies that there must now be a corresponding structure in promise theory: it is *processes* or *work-flows*. Space forces us to refer to this elsewhere [15].

A final possibility of this work is a fundamental redesign of archaic models like the 'CMDB'. Cfengine's answer to the Configuration Management Database (CMDB) is not a traditional inventory system like most present day solutions, but rather a knowledge-based semantic web of information that links database records (occurrences) to manuals, papers and enterprise level policies through the *concepts* they employ. Without the promise concept such a task is very difficult indeed, but using Promise Theory and Topic Maps such a modern information base can now be built.

7 Conclusions

This work shows that there is a two-way mapping between promises and topic maps that enables a simple formal representation of human understanding to be codified. This may be used to annotate policy with meaning and intention, and navigate it efficiently without hierarchical complexity. Using a Promise Theory framework, system policy can be expressed directly in the form of low level *intentions* whose implementation can be promised by cfengine for any initial state of the system. Automation keeps the promises and humans think about why the promises were made – a simple division of labour which makes the best use of each's abilities.

A promise theoretic grounding will always generate a well-formed topic map because the model is directly comparable to that of topic maps. The main difficulty with topic maps is finding a sufficient number of meaningful associations to make a dense enough network for easy 'routing' of thought. Here the lack of a rigid hierarchy is essential, and an effective way to find meaningful trains of thought is to weaken associative logic not simply reason about it as a logical system [15].

The approach proposed here discourages the use of traditional data model management schema, i.e. *inter-occurrence relationships* like *hyperlinks* and the *entity relation* models, and especially object hierarchies. Rather than dealing with hundreds or even thousands of tables in the Common Information Model (CIM), Operational Support Systems or commercial CMDBs, and searching for meaning by brute-force matching of data, we can deal with smaller conceptual neighbourhoods that point more meaningfully from high level intentions to low level implementation. Greater abstraction means fewer things to deal with. Further aspects of this approach will be reported elsewhere.

Acknowledgment: I am grateful to Jan Bergstra, Alva Couch and Steve Pepper for helpful discussions. This work is supported by the EC IST-EMANICS Network of Excellence (#26854)

References

1. J. Strassner. *Handbook of Network and System Administration*, chapter Knowledge Engineering Using Ontologies. Elsevier Handbook, 2007.
2. D. Trastour, C. Bartolini, and C. Priest. Semantic web support for the business-to-business e-commerce lifecycle, 2002.
3. J.M.Serrano, J.Serrat, J.Strassner, and M.O.Foghlú. Management and context integration based on ontologies behind the interoperability in autonomic communications. In *SIWN International Conference on Complex Open Distributed Systems (CODS 2007)*, volume 1, pages 435–442, 2007.
4. J.M. Serrano, J. Serrat, S.V.D. Meer, and M.O. Foghlú. Ontology-based management for context integration in pervasive services operations. In *First International Conference on Autonomous Infrastructure, Management and Security (AIMS 2007)*, volume LNCS 4543, pages 35–48, 2007.
5. S. Pepper. *Encyclopedia of Library and Information Sciences*, chapter Topic Maps. CRC Press, ISBN 9780849397127, 2009.
6. 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*, pages 97–108, 2005.
7. J. Bergstra and M. Burgess. A static theory of promises. Technical report, arXiv:0810.3294v1, 2008.
8. M. Burgess. *The Cfengine reference manual.* http://www.cfengine.org/docs.
9. M. Burgess. Configurable immunity for evolving human-computer systems. *Science of Computer Programming*, 51:197, 2004.
10. M. Debusmann and A. Keller. Sla-driven management of distributed systems using the common information model. In *Proceedings of the VIII IFIP/IEEE IM conference on network management*, page 563, 2003.
11. S. Pepper. The tao of topic maps. In *Proceedings of XML Europe Conference*, 2000.
12. M. Burgess. A site configuration engine. *Computing systems (MIT Press: Cambridge MA)*, 8:309, 1995.
13. M. Brenner, M. Garschhammer, M. Sailer, and T. Schaaf. Cmdb yet another mib? on reusing management model concepts in itil configuration management. In *Proc. 7th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management (DSOM)*, 2006.
14. http://research.iu.hio.no/topicmaps/tm.php.
15. A. Couch and M. Burgess. Compass and direction in topic maps. *(Oslo University College preprint)*, 2009.