

Composing Grid Services through Natural Language

Ian Wakeman David Weir Bill Keller Julie Weeds Tim Owen

May 3, 2005

Abstract

In the Natural Habitat project, we have been investigating techniques for composing pervasive computing services through natural language techniques, and building prototypes to evaluate approaches for the design of the NLP and the middleware. In this paper, we will be examining the possibilities for transferring our approach to the current architectural framework of the Grid, pointing out challenges that need to be overcome on the road to building services that can be composed by people.

1 Introduction

The vision of pervasive computing has often looked to “Star Trek” as an example of how pervasive computing can become embedded in our world, where characters speak and the world operates thus so. Our work has taken this as a goal, and investigated how natural language can be used to construct policies for the customisation and management of the services within a pervasive computing context. Our target users have been normal people within their office or home, and the computing context has been limited to a few services and at most a hundred events per second. Whilst Grid applications are obviously more complex and larger in scope, there are obvious ways in which our approach can transfer to the management and configuration of E-science services. In this paper we describe the current state of our prototype and then discuss to what extent we can transfer our approach to Grid configuration.

2 System Overview

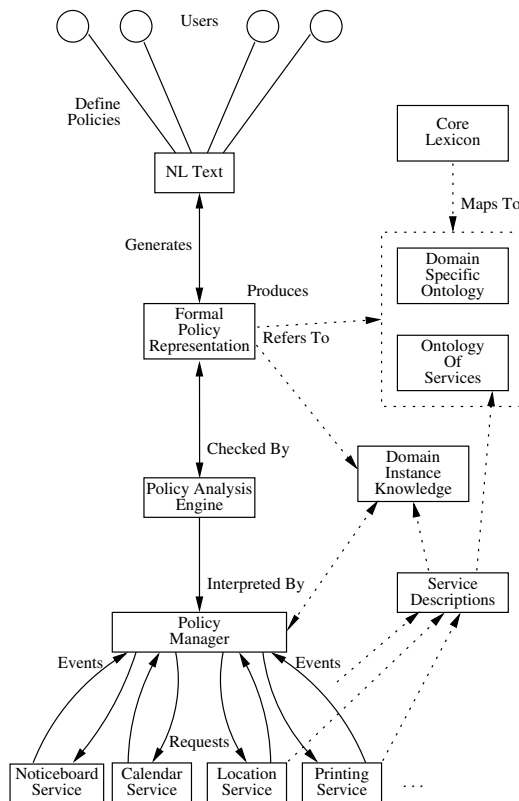


Figure 1: Overview of Policy Management System

Figure 1 shows the high-level architecture of the system we are building. There are a number of software services that provide useful functionality in the user’s environment. These send out notifications of events, and accept requests for action from applica-

tions and other services. The policy manager stands as a central broker for notifications and requests, and creates, modifies or cancels requests in accordance with its set of active policies, before passing them on to the relevant services.

Users can configure and compose the services to personalise their environment by formulating policies that specify what actions should be taken when certain events occur. For example, a user may create a policy such as *Switch the heating down when there is nobody in the house* or another that specifies *Print documents on the nearest printer, by default*.

Our strategy in interpreting the natural language descriptions of policies is to produce a coarse-grained analysis in terms of grammatical dependency relations (e.g., subject, object, modifier) and to interpret this analysis in a way that is driven both by the domain ontology and by the fact that we are anticipating a description of a policy rule. We have found this approach has a number of advantages over making a detailed grammar-based semantic analysis of the user’s policy statement, since coarse dependency analysis neatly captures the redundant structural variation in the language, and allows the logical relations between words available from the ontology to be used to best advantage.

We undertook a series of user studies to investigate how users would like to configure an imagined pervasive environment. From the corpus of results we identified that many configuration policies involve the incremental refinement and rewriting of action requests such as “print colour documents to the colour printer”, whilst the remainder were mostly of the form of conditioned action requests, such as “send me an email when my printer goes offline”.

We therefore designed a policy representation suited to this style of policy. In particular, the representation should be able to cope with the incremental modification of requests for action, including:

- the ability to partially specify how requests will be satisfied;
- the ability to fill in default constraints if they are not already specified;
- the ability for new constraints to override and

have preference over previously specified constraints.

The general form of a user policy is a rule with a precondition and postcondition, each expressed using description logic terms that reference concepts, roles and individuals in the ontology. For example, a user’s policy “*Print colour documents on a colour printer*” would be expressed formally as:

$$x \in \text{Print} \sqcap \text{patient.colourness.COLOUR} \Rightarrow x \in \text{target.colourness.COLOUR}$$

where **Print** is a concept name of a request to print, which has roles called **patient** and **target** (which in turn each have **colourness** roles). This policy rule can be read as follows.

If there is a **Print** request whose **patient** (i.e., the document to be printed) has the value **COLOUR** for its **colourness** role, then assert the postcondition that the **target** of the request must be a **Printer** whose **colourness** role has the value **COLOUR**.

However, **modification** policies, which involve incremental refinement of existing action requests, come in two distinct flavours. Some user policies, which we refer to as **overwrite** policies, have a postcondition that in effect contradicts the preconditions of the rule. For example given the policy “*If a document is sent to printer lja and it is offline, send to lja instead*” the original target of the print request is changed by the new assertion in the postcondition. In contrast, other user policies, which we refer to as **default** policies, have a postcondition which is only asserted if it does not contradict existing knowledge about the request. This captures policies such as “*Print documents on the nearest printer, by default*” where we do not overwrite existing target information, but only assert it if it is not specified. In order to distinguish these types of modification policy, each rule is assigned a tag that indicates whether overwrite or default semantics is intended.

The other class of policies, referred to as **trigger** policies, are completely distinct from modification policies and involve a postcondition that creates a completely new request, rather than asserting

changes to some existing event. For example, the policy is “Alert me if someone prints on my printer” can be captured in the following rule:

$$\begin{aligned} x \in \text{Print} \sqcap \text{target.owner.TIM} &\Rightarrow \\ y \in \text{Alert} \sqcap \text{recipient.address.‘alert@me.com’} & \end{aligned}$$

where the use of distinct variables x and y is taken to indicate that different requests are being referred to.

Note that we are still able to represent traditional deontic-style policies relating to authorisation and obligation. In general, these can be captured using a modification policy with overwrite semantics. For example, “Bill is not allowed to print” is expressed as

$$\begin{aligned} x \in \text{Print} \sqcap \text{agent.name.BILL} &\Rightarrow \\ x \notin \text{Print} & \end{aligned}$$

The task of the policy manager is to enforce these policies, by acting upon incoming events such as notifications about person-movements from the location service or print requests from an application. The manager may modify a request (e.g. naming a target printer if not specified) or initiate a new request (e.g. asking the heating service to lower its setpoint) as required to implement the user’s policies. Currently, we have built a prototype system of most of the major components shown in Figure 1. The knowledge base module is implemented using the RACER system [1] which provides a DL reasoning engine, as well as storing the definitions of domain concepts and roles (the TBox) and the known individuals (the ABox).

3 Application to the Grid

The above system was designed for small computing contexts, where the number of services and consequent events and requests were relatively low, and all services were under control of a single management domain. What aspects of the system can we transfer across to the Grid?

We believe that our approach to the composition of services through descriptions in natural language has many possibilities as an interface technique. The

grid is used by domain experts, who have a deep understanding of the interactions between the services and events and of the specialised language used in the domain, but who may not be proficient in exposition by programming. As more domain specific ontologies are constructed, encouraged by initiatives such as the Semantic Grid [2], the natural language problem becomes more tractable. The use of distributional similarity is a very promising approach for disambiguating language, where large corpuses of text are collected, and then used in the statistical analysis of phrases. The use of meta-descriptions to annotate the collected text will improve the efficiency of this approach.

The patterns of service composition supported in our current work can obviously be of use in customising interacting services, eg to trigger the collection of data when threshold conditions are surpassed, and in controlling how different services use shared resources such as network bandwidth. We are currently investigating whether new services can be constructed by a process of concept definition - eg the concept of “nearest” can be defined by the logical combination of a set of different location-related services. Such approaches hold promise for e-scientists to construct reusable services across research projects.

However, there are major problems with scaling the services and events. The prototype described above directs all events through a single policy manager. Further, the state of the computing context is mirrored within the ABox of the DL engine. When thousands of events occur per second, these approaches will not scale, as in any complex management situation. Possible avenues to explore for solutions would be in developing layers of abstraction, so that the NL policies were concerned with less frequent events, or in distributing policies. Aggregation of events into more abstract events is often used within network management systems, and might be an appropriate area for investigation of the advantages of mobile code - see for example the Sahara system [3]. Distribution of policies would require analysis and decomposition of policies into sub-goals and sub-policies. This is in many ways the holy grail of policy-based management, and there is a raft of work within the policy management community - see for example

Kephart's vision of autonomic computing [4]. An alternative approach maybe in the construction of interface tools which constrain the user to describing only feasible policies, or by restricting the user to on;y working meta-descriptions of already composed services.

For the configuration of services to be automatic, there are additional pieces of infra-structure required. How can expert service construction be saved and re-used? How best to judge which services can work together? How to develop middleware which is oriented to the development of ontological construction? We need both ways to describe interaction and construction, and a repository to save this meta-information. The Semantic Web [5] and the underlying description logics [6] are an obvious lead to follow.

semantic web. In *Lecture Notes in AI*. Springer, 2003.

References

- [1] Volker Haarslev and Ralf Mller. Description of the racer system and its applications. In *Proceedings International Workshop on Description Logics (DL-2001)*, Stanford Ca, August 2001.
- [2] N.R. Shadbolt N.R De Roure, D. Jennings. The semantic grid: Past, present, and future. *Proceedings of the IEEE*, 93(3):669–681, March 2005.
- [3] Bhaskaran Raman, Sharad Agarwal, Yan Chen, Matthew Caesar, Weidong Cui, Per Johansson, Kevin Lai, Tal Lavian, Sridhar Machiraju, Z. Morley Mao, George Porter, Timothy Roscoe, and Mukund. The sahara model for service composition across multiple providers. In *Proc. of International Conference on Pervasive Computing*, Zurich, Switzerland, August 2002.
- [4] Jeffrey Kephart and David Chess. The vision of autonomic computing. *IEEE Computer*, pages 41–50, January 2003.
- [5] T. Berners-Lee, J. Hendler, and O. Lassila. The semantic web. *Scientific American*, 284(5):34–43, 2001.
- [6] Franz Baader, Ian Horrocks, and Ulrike Sattler. Description logics as ontology languages for the