

## **myGrid WP4 till All Hands 03, suggested plans**

<b>Document Ref:</b>	myGrid WP4 till All Hands 03, suggested plans
<b>Author:</b>	Phillip Lord
<b>Date:</b>	July 14, 2003
<b>Pages:</b>	7
<b>Abstract:</b>	This document suggests a number of tasks that myGrid WP4 could focus on before the All Hands Meeting in September, which is the next “demo” that we have

## Contents

<b>1</b>	<b>Introduction</b>	<b>3</b>
<b>2</b>	<b>Components</b>	<b>3</b>
<b>3</b>	<b>The Reasoner and DIG</b>	<b>4</b>
<b>4</b>	<b>DAML+OIL API's</b>	<b>4</b>
<b>5</b>	<b>Instance Store</b>	<b>4</b>
5.1	Versions . . . . .	4
5.2	A hands free instance store . . . . .	5
5.3	Netbeans Instance Stores . . . . .	5
<b>6</b>	<b>Find Service</b>	<b>5</b>
6.1	Multiple instances . . . . .	5
6.2	More Widespread Searching . . . . .	5
6.3	A query interface . . . . .	6
6.4	Subscription Interface . . . . .	6
<b>7</b>	<b>Misc</b>	<b>6</b>
<b>8</b>	<b>MIR interactions</b>	<b>6</b>
8.1	General Optimisation . . . . .	7
8.2	Hand over Service Browser . . . . .	7

## 1 Introduction

This document is not meant to be a formal road map, but a “bag of ideas” of things that we could do before the All Hands Meeting. I’ve tried to give some ideas of how long each task would take to address. Some of the tasks depend on, or would be made substantially easier, others.

Here I am just considering those parts of WP4 that I have done, which mostly relates to the discovery of descriptions, rather than providing the descriptions which has mostly been Chris’s work. I haven’t covered this because I don’t know it in great enough detail (something which I probably need to change!).

In the first section I describe the current components and then in following section work through them in order. Probably this is roughly the order they should be tackled in, as it will isolate later tasks from changes further down the line, although this is not a rigid requirement. The final section is the “misc” section.

## 2 Components

The current components, which WP4 has direct, or within Manchester, control of, are as follows:-

**The Reasoner** Currently either RACER or FaCT, which we are addressing through the DIG interface.

**DAML+OIL API’s** There are a large number of syntactic transformations going on, all of which use API’s culled rather inelegantly from OilEd.

**The Instance Store** We are using the Instance Store as our main access to the reasoner. The main theoretical advantage of this is that it means we escape A-box reasoning, which is known to be hard. My main reason for using it however, is that it means we need only read-only access to the reasoner, when asserting descriptions of services, which should lessen concurrency, and security issues.

**The Find Service** The find service is an interface to the instance store, through which services descriptions are registered and can be discovered. This consists of a client, from which we discover services, and a indexer, which gathers the descriptions for which ever component is providing them.

**The Service Browser** Allows viewing of services in a registry view, and provides an example case of using the find service.

Additionally there are a number of other components outside our direct control.

**Registry Views** The place we are getting out service descriptions from.

**Netbeans** The IDE we are building within.

**MIR** I have no current interaction with this, but probably should do. Chris’s registration stuff does have interaction with this, but probably should not do.

### 3 The Reasoner and DIG

**Background:** DIG is a generic interface to reasoners. All communication is via a SOAP-like set of XML messages. Both RACER and FaCT support this interface.

**Problem:** Generally there aren't any problems with this. We can just use it. Mostly I am using this through tomcat because of problems with the JLinker interface (the link between FaCT, and the DIG client code) on linux meaning I have to run two versions of Java. This also allows us to separate the reasoner and the client, which as <sup>my</sup>Grid is a distributed project, is good. The whole bundle tends to be fairly slow however. Currently FaCT limits us to Windows or Linux (not Solaris), which is a pain. There are no problems associated with DIG that I am currently aware of.

**Solution:** FaCT is being rewritten anyway. Hopefully FaCT-2 should have a DIG interface attached soon, and as its being written in C/C++ we should be able to bolt this on using an JNI interface, which should be quicker than JLinker, and should actually work. For all these reasons we should probably start using FaCT-2 as soon as its ready, even if we get some crashes as its early software. This should mostly be a "runtime" decision. We can switch back easily. Likewise with DIG, we should probably trace version changes as and when they happen.

**Duration:** A couple of days, as we will almost certainly hit some unexpected bugs.

### 4 DAML+OIL API's

**Background:** We currently don't really have any DAML+OIL API's per se, but rather have the data structures that Sean wrote for OilEd. We do have a set of all new OWL data structures, which were written partly with <sup>my</sup>Grid resources (i.e. me and Angus), and which are probably nearly ready to go.

**Problem:** The existing DAML+OIL API's bring with them a lot of dependency (as we get all of OilEd at once). <sup>my</sup>Grid has a serious dependency problem anyway. On the flip side the DAML+OIL API's are well tested, and reasonably stable. We need to start building language specific support in to the find service (or at least some related module) to allow us to build and express queries easily. This is currently the worse aspect of the semantic support in <sup>my</sup>Grid ... queries are built by concatenating RDF/S XML strings together in Java, which is really bad.

**Solution:** I'm really not sure about this at all. I don't want to build language specific tools into the find service based on the DAML+OIL data structures, if I am going to have to take them out again. But its also new software, and there are going to be problems. Also Chris's software using the DAML+OIL software more heavily than mine does, and will require some re-writes.

**Duration:** Probably a couple of weeks if we choose to go with the OWL support.

### 5 Instance Store

#### 5.1 Versions

**Background:** The instance store is produced by Daniele, who seems to be extended it rapidly at the moment.

**Problem:** I got Daniele to checkpoint the code about a month before the IF. This gave us something stable to work against. We need to update to the current version though, or any bugs we find will end up being our problem, rather than Daniele's

**Solution:** We should update ASAP.

**Duration:** 1 day, to understand the changes in the API's.

## 5.2 A hands free instance store

**Background:** The instance store is wrapped around a “backing store” and a reasoner. The reasoner I have discussed in Section 3. The backing store can be implemented through an interface, but currently all of Daniele's interfaces are RDBMS based.

**Problem:** The SQL interface to the backing store means the system needs an RDBMS, which needs administration. Worse it needs one per user. It's a pain, and as we currently have few services its unnecessary.

**Solution:** Either I need to implement an alternate backing store implementation to work in memory over Java hashes. I did this for an early implementation, but it looks more complex now. The quickest solution is to use an in memory Java RDBMS like hypersonic.

**Duration:** 2 or 3 days.

## 5.3 Netbeans Instance Stores

**Background:** The instance store is currently importantly into Netbeans within the find service.

**Problem:** Changes in the instance store percolate quickly into the find service. Its also hard to find out what the instance store is actually doing, what data it has, and what the reasoner is doing.

**Solution:** Import the instance store as a Netbeans module, and provide configuration within this module. We also need GUI tools to see what is going on. We can hopefully use existing JDBC support modules within Netbeans for the backing store, and steal most of the Daniele's test GUI for the rest. This is particularly important for the hands free instance store (see Section 5.2) as we will no longer be able to directly query the database.

**Duration:** Up to a week (assuming I do this first. A lot of this time will be the netbeans stuff).

# 6 Find Service

## 6.1 Multiple instances

**Problem:** We can't search over more than one view. There can only be one instance of the find service, at the moment. Additionally there can only be one view (the URL of which is hard coded!).

**Solution:** This is largely a simple plumbing issue, although if each find service requires an instance store of its own, I need to be able to generate new databases, which may not be trivial, unless I have a in memory database.

**Duration:** 1 week

## 6.2 More Widespread Searching

**Background:** I wrote the find service to operate over the view, so that currently it only discovers services. The main reason for this was trivial. We got the views software earlier, and its easy to set up.

**Problem:** During IF4 we hacked things to make it also discover workflows in the MIR, but this really was a hack. Chris put the location of the workflow in the MIR into the metadata in the view. Really it would make sense for the find service to be capable of searching the MIR.

**Solution:** The find service currently works over a thin abstraction called a “FSRegistry”. An adaptor between this and the View is what makes this all work (This FSRegistry supports a “notification” event based interface, which the view doesn’t, so this adaptor polls for service addition or removal). Assuming that the MIR has nice client software (and the demo and its use within the workbench suggests that it has, or is at least heading that way), it should be possible to search over the MIR, for workflows, or data items.

**Duration:** Very difficult to judge, as I haven’t started to look at the API. A couple of weeks minimum, I would guess though. Additionally the MIR API will change with time, so there is an ongoing maintenance.

### 6.3 A query interface

**Background:** The find service supports queries in DAML+OIL, which need to be generated in some way.

**Problem:** We have no way of forming these queries. During IF4 I had a nasty hack, which was an RDF based string catenation, which is not tenable even in the extreme short term.

**Solution:** Not sure. I have several ideas. I could use Chris’s “growl” compiler. Or I could use either the DAML+OIL, or the OWL API’s.

**Duration:** Unsure. I don’t have a clear idea of what sort of queries I would want to provide at the moment, although what I have now, but better would take a week.

### 6.4 Subscription Interface

**Background:** The current interface is a query-response interface. “What things match this?”.

**Problem:** This interface is supports the ServiceBrowser “query by navigation” UI fairly badly.

**Solution:** Provide a query subscription interface. The alternative to this is some sort of polling at the UI level. I think that Netbeans supports this, but I’m not sure at the moment.

**Duration:** I’m not sure how to do this at the moment. I think it needs enhancements to the instance store interface. A couple of weeks at least, a lot more if I have to do the instance store enhancements.

## 7 Misc

## 8 MIR interactions

**Problem:** The find service, and/or the service browser manages to half kill the MIR. Its already slow. The service browser really doesn’t help, even though the two do not interact, or rather shouldn’t interact.

**Solution:** Probably this will just go away as we change other things. If it does not, then I think I know what is causing it, but, of course, I can not be sure. This needs to

be solved at some time, before All Hands, but exactly when is not clear. If I do know work with the MIR, then there as late as possible. If I do need to work with the MIR, it will slow the development process.

**Duration:** Probably a day, but maybe a week!

## 8.1 General Optimisation

**Background:** The system is really really slow.

**Problem:** Hack code. The interaction with the view (which is not fast to start off with), is really poor, and heavily duplicative.

**Solution:** Again much of this will come out in the wash. Judicious use of threading will enhance the user experience.

**Duration:** Between one day, and five weeks. . .

## 8.2 Hand over Service Browser

**Background:** The service browser is really about viewing services, so it would appear to be Soton's domain. Luc has expressed a desire to take it over. But the semantics stuff clearly needs support from us.

**Solution:** The Service Browser gets given to Soton, and we re-work the semantics as a "view" thing. . . much like sort by date, or whatever. gg

**Duration:** Probably a week. The service browser code needs to be dumped entirely, as it needs a total re-write. Hopefully Soton will do much of this, but I will need to work with them, to ensure that I can add the semantics option.