An Agent-based e-Science Experiment Builder

Christopher D. Walton[*] and Adam D. Barker

Centre for Intelligent Systems and their Applications (CISA),
School of Informatics, University of Edinburgh, Scotland, UK.
Email: `cdw@inf.ed.ac.uk` Tel: +44-(0)131-650-2718

Abstract. In this paper we demonstrate the use of agent technology to assist in the construction and enactment of e-Science experiments. Our approach is founded on the adaptation of an agent protocol language to perform composition of web services. We present a definition of the language, and show how it can be used to express e-Science experiments. We also describe a tool, called MagentA, which allows experiments to be rapidly constructed, verified, and enacted.

1 Introduction

e-Science may be defined as the use of distributed electronic resources, by scientists, to solve scientific problems. There is a continuing programme of work on the construction of an infrastructure, called the *Grid* [8], for supporting the kind of wide-scale distributed computing required for e-Science. Similarly, there is a concurrent programme of work on the construction of software technology for the specification and execution of e-Science experiments. The specification of e-Science experiments has evolved from early ad-hoc methods to the use of *workflows* [7]. At the same time, the execution of e-Science experiments has moved from parallel programming techniques to the use of distributed web technology, in particular *web services* [2]. Nonetheless, there remain significant challenges to be addressed in the construction of e-Science experiments, particularly in relation to the composition and inter-operability of services, and a solution to many of these is still far from clear.

The *composition* of services is fundamental to the way in which e-Science experiments are constructed from distributed resources, i.e. services. It is intended that a scientist will be able to simply and rapidly compose these services into systems which define experiments. Composition works at two different conceptual levels; the *workflow level*, and the *enactment level*. Workflows, which are experiment specifications, are constructed by the scientist through an appropriate interface. At the workflow level, the composition is performed in terms of black-box services which are assembled according to the experiment plan. For example, the workflow illustrated in Figure 1 of this paper describes the Brightest Cluster Galaxy experiment, taken from the AstroGrid project [1]. It should

[*] This work is sponsored by the EPSRC Advanced Knowledge Technologies (AKT) Interdisciplinary Research Collaboration (Grant GR/N15764/01).

be noted that there is currently little consensus on the kind of workflow model which is most appropriate for the specification of e-Science experiments, and there are many proposals including data-flows, business-process models (BPM), and UML activity diagrams. In this paper we are primarily concerned with the enactment level, and we simply adopt the UML activity diagram formalism for workflow specification.

In order to enact an experiment, it is necessary to perform a *workflow mapping* from the abstract experiment workflow to an executable system, composed of executable services. We note that there is not necessarily a one-to-one map between services at the workflow level, and services at the enactment level. The enactment of a single workflow service may result in a complex execution behaviour. Furthermore, there is a strong desire to conceal the mapping task from the e-Scientist. However, we note that workflow mapping is a complex task and remains an open problem. For this paper, we assume that mapping will be performed by manual refinement of the experiment specification. At the enactment level, we are interacting with physical services (e.g. telescopes), computation resources (e.g. databases), and humans. For convenience, we make the standard assumption that these kinds of services can all be represented and controlled though a *web service* interface. For example, complex equipment can be controlled though the invocation of web services, and interaction with humans can be performed abstractly via web services by means of web pages and forms. Thus, we reduce the complexity of the problem to one of web service composition.

The composition of web services into experiments that can be enacted, requires a high degree of *interoperability* between services. It is necessary that services built by different organisations, and using different software systems, are able to communicate with one another in a common formalism with an agreed semantics. To address the inter-operability issue, technology from the semantic web is increasingly being used in the specification and execution of e-Science experiments. For example, the use of semantic meta-data in the Grid is described in [11]. This has lead to the promotion of the term *Semantic Grid* which envisions a convergence between the Grid and the Semantic Web [4]. For the purpose of this paper, we will assume that the services we use in our experiments are known in advance. There are still many unsolved issues relating to automatic discovery and invocation of web services that we do not address. These issues are currently the focus of initiatives such as OWL-S [3], and WSMF [6].

It should be noted that the semantic web itself is closely related to the field of agency, and many ideas from the semantic web are derived from multi-agent systems. Indeed, the semantic web itself is predicated on the notion of *agents* as the key consumers of semantic web information. We continue this theme in our paper by showing how agents can address the issues of composition and interoperability among web services in experiments. If we consider the scenario in Figure 1 we can observe that there is a significant degree of coupling between the services in the workflow. Consequently, in any reasonable mapping from this workflow to web services, there will also be a degree of interaction between the different web services in the resulting system. We must therefore consider how we can represent this interaction in our system. At this point we encounter the limitations of current semantic web and web service technology.

Interaction between web services is currently accomplished by a remote procedure call (RPC) mechanism, by the exchange of SOAP messages between web service clients and containers. The SOAP specification defines a one-way stateless communication mechanism, but this is too restrictive for our purposes. In order to define experiments, we need to define complex communication patterns between services, for example, broadcast or multi-cast communication. An additional problem concerns how we compose the web services which comprise an experiment. It is possible to construct such a system through a static composition of services, where the composition is encoded directly into the services. However, this approach is error-prone and inflexible as it does not allow us to easily change the kind of experiment we define. Ideally, we would like a separate representation which can express complex patterns of interaction between web services, such as we describe. We are aware of the activities of the W3C Web Services Choreography group on defining a suitable representation for performing choreography (i.e. composition) between web services, but we are unaware of any implementation of their proposals.

The focus of this paper is on the definition of a formalism for the realisation of interaction between web services, taking our previous work on multi-agent protocol languages as our inspiration [10]. Our script-based representation for representing multi-agent protocols can be readily adapted to express coordination between web services [9], and therefore provide a suitable model of composition. We designed our protocol language to be independent of the rational processes of the agents, and therefore we require only minimal changes to the language to make it applicable to web services. Our language appears to be a good complement to SOAP, as both are independent of the message content or implementation. We also describe a platform which we have built, based upon our protocol language, for defining and enacting e-Science experiments.

Our presentation in this paper is structured as follows. In section 2 we describe the Brightest Cluster Galaxy scenario which we use as a motivating example for our technique. Subsequently, in section 3 we define a composition framework for specifying and enacting e-Science experiments. Our framework comprises both the MAP language (Section 3.1), and the MagentA tool (Section 3.2) which we define in detail. Finally, we conclude in Section 4 with a discussion of our future work.

2 AstroGrid Example Scenario

Our example scenario is based around the location and comprehension of the properties of "Brightest Cluster Galaxies". If one observes clusters of galaxies with a range of sizes/luminosities, it is often apparent that there is one galaxy which is much brighter than all the others. This galaxy, called the Brightest Cluster Galaxy (BCG) is frequently positioned in the centre of the cluster. Statistically, it can be shown that the BCG is something more than just the brightest galaxy in the cluster: galaxies in clusters follow a fairly general distribution of luminosities, and BCGs occur far too often and are far too bright to be simply the upper end of that distribution. They are real outliers, pointing to some different process of formation and/or evolution. The scientific background to this scenario is that there is some evidence for correlations between the properties of the BCG and of the cluster of galaxies in which it resides. This indicates that the cluster is affecting the way that the BCG is formed or has evolved, but scientists don't yet know how this works. Our example scenario, expressed in Figure 1, envisages an astronomer trying to discover how BCG are formed.

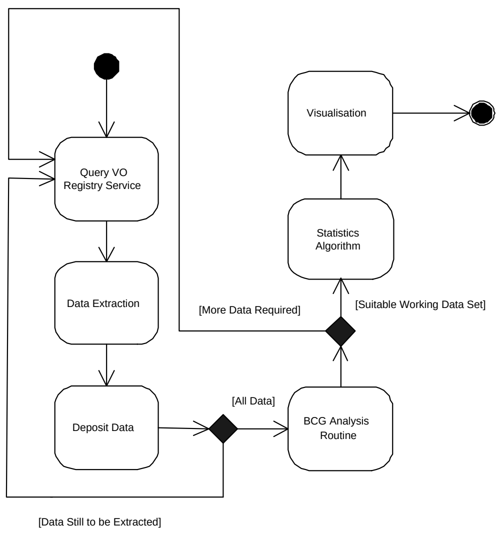


Fig. 1. Brightest Cluster Galaxy Workflow

We suppose that our astronomer has a hypothesis about connections between the properties of BCGs and those of their host clusters. In the first step of our experimental workflow, the astronomer performs a query on the Virtual Observatory (VO) to construct a sample of cluster/BCG pairs which have been well observed in a number of pass bands. To achieve this, the astronomer queries the VO Registry web service and obtains a list of VO data sources which are classified as being catalogues of clusters of galaxies. This yields a list of cluster catalogues; some based on optical/near-IR observations, some on X-ray observations, and some on sub-millimetre observations. Examples of real astronomy data sources include: the XMM-Newton Science Archive (X-Ray), the Sloan Digital Sky Survey (SDSS) and the UK Infrared Deep Sky Survey (UKIDISS). In the second step of the workflow, the VO data sources are individually contacted in order to extract the positions of all of the clusters included in each catalogue. The web service for each of these data sources is accessed in order to extract all the attributes of all sources contained in a search radius of a certain size around the position of each of the clusters. In the third step of the workflow, the collected data is deposited in the AstroGrid MySpace storage facility. The VO Registry Service may be repeatedly referenced (as indicated by the loop in the workflow), for data sources which are classified as catalogues of optical, X-ray, near-infrared or radio sources (and therefore may include relevant observations of BCGs).

In the next stage of the workflow, an analysis routine is performed on the data which has been deposited in MySpace. The analysis routine has to work out which galaxies in the galaxy catalogue data are the BCGs in each of the host clusters and generate a combined set of all the data known about each Cluster/BCG pair. Not every BCG/cluster pair has a value for each attribute, but most have values for the great majority of them, so this is deemed to be a good working data set. So for each cluster in the catalogue there is likely to be a number of properties recorded; obvious things like position, brightness, size, etc. There will also be another set of attributes recording properties of the sources in some optical or near infrared catalogue. At this point the scientist may determine that more data is required, and so the workflow includes a link back to the beginning. Alternatively, the astronomer may run a statistics algorithm, which seeks the attributes with the highest information content on the deposited lump of data. The outputs are then fed into a graphics package which generates a grid of scatter plots for pairs of attributes, arranged in order by the strength of correlation between them. If there are N attributes for M BCG/Cluster pairs then the Grid of Scatter plots represents $N*(N-1)/2$ plots, each with M points plotted. In other words each attribute is plotted against each other attribute for the set of BCG/Cluster pairs. In the final stage of the workflow, a visualisation allows further investigation into the correlations.

We will now consider the mapping between the workflow that we have defined and the actual services which will be used during enactment. Figure 2 shows the different services that are used, and the interactions between them. It is clear that there are significant similarities between the services and the workflow, though we note that this will not always be the case. A surprising feature of the mapping, is that the scientist is explicitly represented by a service during enactment. This is necessary as we must define precisely how the scientist interacts with the services, and it is necessary for this service to supply an appropriate interaction mechanism with the human scientist, e.g. web forms.

There are four main services that are used during enactment, which loosely correspond to the tasks defined in the workflow. In the workflow, the BCG analysis and statistics algorithm were defined separately, as we wished to indicate a decision between them. However, both activities are enacted by the same BCG Analysis service. Similarly, the Data Deposit task is performed as part of the Extraction service. The key difference between the workflow and the enactment is that we must also consider the interactions with the scientist, the external data sources, and the MySpace data storage facility.

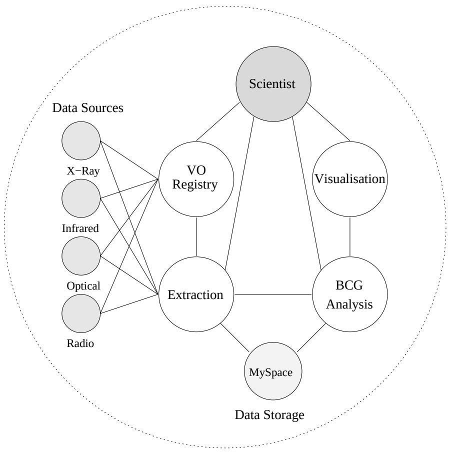


Fig. 2. BCG Service Composition

The pattern of interaction between services in our example experiment is indicated by the connections between services in Figure 2. We note that this is just one possible way of composing the services, and alternative compositions are also possible. Furthermore, we do not define the actual interaction sequence, which will be complex and non-deterministic. We note that a *static* composition of services, where the pattern of interaction is directly encoded into each of the services, is certainly possible. However, this approach is inflexible, and restricts the services to a single kind of interaction, or a single experiment. In this paper, we adopt a *dynamic* approach, which affords considerable flexibility in the composition of services.

3 A Composition Framework

In order to construct a framework for dynamically composing web services into experiments, we consider how composition can be applied to an arbitrary collection of web services. We present a language for expressing composition later in this section. However, we first consider such a language will be disseminated and enacted. A bit of though leads us to the approach shown in Figure 3. In order to coordinate a group of web services, we can equip each web service with some extra functionality, which we term the *agent stub*. This stub is responsible for enacting the coordination protocol expressed in our language. We can disseminate the protocol to the individual web services at the start of an experiment, and then the enactment of the protocol by the agent stub on each service will supply the required coordination. However, while this approach is plausible, we adopt an alternative which we believe is better suited to heterogeneous systems. In particular, there are two significant issues which relate to the use of stubs.

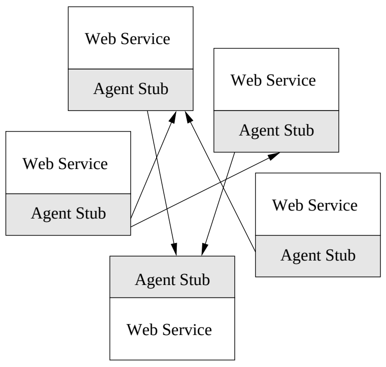


Fig. 3. Coordinating Web Services with Agent Protocols

The first issue is a direct result of the modification of the web services. We are in essence converting web services into a kind of agent and thereby breaking compatibility with existing web services and web service architectures. Any web service participating in an experiment must be modified. However, we do not necessarily have control over all of the services which we utilise in an experiment. Therefore, it appears infeasible for any approach to require the modification of all services before coordination between services can be performed. Ideally, we would like an approach to coordination which does not require modification of the web services involved.

The second issue concerns the implementation of the agent stubs which provide the coordination. In the general case, these stubs would provide a common set of functionality to the web services and be essentially identical in functionality. However, web services may be implemented in a variety of different languages and this would require the agent stub to be reimplemented in the language of the web service. Ensuring that the agent stubs implemented in different languages provide the same functionality and are completely compatible is a non-trivial task. Again, it is clear that the modification of the web services will create significant issues in this approach.

Our chosen approach, illustrated in Figure 4, does not break compatibility with the existing web service architecture as the only kinds of components we define are web services. The agents (labelled A) in the diagram are analogous to the agent stubs in the previous method. However, the agents are composed into a close-coupled system, which itself resides within a web service. Each agent A acts on behalf of a single external web service. In effect, we define a coordination mechanism which is entirely external to the web services which are coordinated, and the web services do not need any knowledge that they are participating in a coordination. This has significant advantages in that the web services do not need to be modified, and the protocol does not need to be disseminated between services. An additional advantage is that this technique reduces the communication cost of the coordination. A significant amount of communication

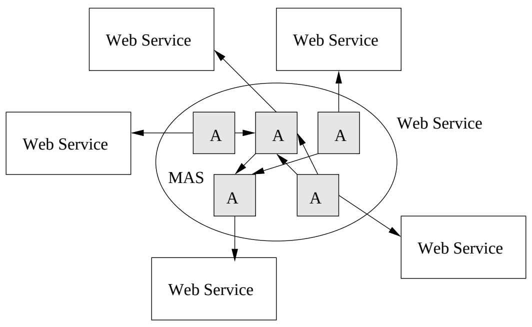


Fig. 4. Alternative Coordination Method

now occurs within a single web service, rather than between distributed web services. External communication is only performed when an agent needs to invoke a method on a particular web service.

Having now defined an appropriate method of coordination, we will turn our attention to the definition of the language for performing coordination. We return to the implementation details at the end of the section. As stated in the introduction, we adopt a coordination technique which we previously defined for coordinating web services in multi-agent systems. This technique is expressed by our Multi-Agent Protocol (MAP) language, which is formally defined in [10]. The adaptation of the language to the coordination of web services requires very few changes. In our chosen technique, the language is in essence still defining a coordination among agents (labelled A in Figure 4). The agents are now acting as proxies for external web services, rather than having behaviours of their own. The only real change to the language is a mechanism for expressing the invocation of web services. Given the straightforward nature of the changes, we retain the MAP name for our language.

3.1 MAP Language Definition

MAP is a lightweight dialogue protocol language derived from process calculus. We note that MAP is only intended to express protocols, and is not intended to be a general-purpose language. Therefore, the relative sparsity of features in the language, e.g. no user-defined data-types, is appropriate. Furthermore, MAP is designed to be a lightweight language and only a minimal set of operations have been included. It is intended that MAP protocols will be automatically generated, e.g. from a planning system or a workflow enactment engine, rather than being constructed by hand. Thus, although MAP protocols appear complex, an e-Scientist would not be required to understand them in detail. MAP can be viewed as a replacement for the state-chart representation of protocols found in Electronic Institutions (EI) [5]. MAP provides an executable specification of agent protocols, while retaining the concepts of *scenes*, and *roles* found in EI.

The division of agent dialogues into *scenes* is a key concept in MAP. A scene can be thought of as a bounded space in which a group agents interact on a single task. The use of scenes divides a large protocol into manageable parts. Scenes also add a measure of security to a protocol, in that agents which are not relevant to the protocol are excluded from the scene. This can prevent interference with the protocol and limits the number of exceptions and special cases that must be considered in the design of the protocol. We assume that a scene places a barrier on the agents, such that a scene cannot begin until all the agents have been instantiated, and the agents cannot leave until protocol enactment is complete.

The concept of an agent *role* is also central to our definition. In MAP, each agent is identified by both a name and a role. Agents are uniquely named, but must be assigned a role which is specified in the protocol. The role of an agent is fixed until the end of a scene, and determines which parts of the protocol the agent will follow. Agents can share the same role, which defines the agents as having the same capabilities, i.e. the same web service interface. Roles are useful for grouping similar agents together, as we do not have to specify a completely separate protocol for each individual agent. For example, we may wish to interact with a large number of different database services, all with the same interface. We can simply define a single role (and associated protocol) which corresponds to a generic database access service, rather than defining a separate protocol for each service. Roles also allow us to specify multi-cast communication in MAP. For example, we can broadcast messages to all agents of a specific role.

$$
\begin{array}{lcll}
P \in \text{Protocol} & ::= & n(r\texttt{\{}\mathcal{M}\texttt{\}})^{+} & \text{(Scene)} \\[4pt]
M \in \text{Method} & ::= & \texttt{method}(\phi^{(k)})\ \texttt{=}\ op & \text{(Method)} \\[4pt]
op \in \text{Operation} & ::= & \alpha & \text{(Action)} \\
 & | & op_1\ \texttt{then}\ op_2 & \text{(Sequence)} \\
 & | & op_1\ \texttt{or}\ op_2 & \text{(Choice)} \\
 & | & op_1\ \texttt{par}\ op_2 & \text{(Parallel)} \\
 & | & \texttt{waitfor}\ op_1\ \texttt{timeout}\ op_2 & \text{(Iteration)} \\
 & | & \texttt{invoke}(\phi^{(k)}) & \text{(Recursion)} \\[4pt]
\alpha \in \text{Action} & ::= & \epsilon & \text{(No Action)} \\
 & | & \phi^{k}\ \texttt{=}\ p(\phi^{l})\ \texttt{fault}\ \phi^{m} & \text{(Procedure)} \\
 & | & \rho(\phi^{(k)})\ \texttt{=>}\ \texttt{agent}(\phi_1,\ \phi_2) & \text{(Send)} \\
 & | & \rho(\phi^{(k)})\ \texttt{<=}\ \texttt{agent}(\phi_1,\ \phi_2) & \text{(Receive)} \\[4pt]
\phi \in \text{Term} & ::= & \_\ \mid\ a\ \mid\ r\ \mid\ c\ \mid\ v &
\end{array}
$$

Fig. 5. MAP Abstract Syntax.

We will now define the abstract syntax of MAP, which is presented in Figure 5. We have also defined a corresponding concrete XML-based syntax for MAP which is used in our implementation. However, we restrict our attention in this paper to the abstract syntax for readability. A protocol P is uniquely named n and defined as a set of agent roles $\mathcal{A}$, each of which defines a set of methods $\mathcal{M}$. A single method M takes a list of terms $\phi^{(k)}$ as arguments (the initial method is specified by an empty list of arguments). Agents have a fixed role r for the duration of the protocol, and are individually identified by unique names a. Protocols are constructed from operations op which control the flow of the protocol, and actions α which have side-effects and can fail. Failure of actions causes backtracking in the protocol.

The interface between the protocol and the external web service, is achieved through the invocation of procedures p. A procedure is parameterised by three sequences of terms. The input terms ϕ^l are the input parameters to the procedure, and the output terms ϕ^k are the output parameters, i.e. results, from the procedure. A procedure may also raise an exception in which case the fault terms ϕ^m are bound to the exception parameters, and backtracking occurs in the protocol. Interaction between agents is performed by the exchange of messages which are defined by performatives ρ, i.e. message types. The parameters to procedures and performatives are terms ϕ, which are either variables v, agent names a, role names r, constants c, or wild-cards `_`. Variables are bound to terms by unification which occurs in the invocation of procedures, the receipt of messages, or through recursive method invocations.

It is helpful to consider an example protocol in order to obtain an understanding of the protocol language. We therefore present a MAP protocol for the data extraction phase of our BCG example in Figure 6. For brevity, we do not present the protocol for the other services in this paper. We distinguish between the different types of terms by prefixing variables names with `$`, and role names with `%`, and we write `e` for an empty (ϵ) action. We use type abbreviations `A` for agent, `R` for role, `S` for string, and `L` for list. We define a single `%extraction` role which expresses the extraction protocol. The role will be enacted by an agent which is a proxy for an external extraction web service. The procedure calls in the protocol, e.g. `extractNext`, are mapped to the associated web service.

The extraction protocol shown in Figure 6 proceeds as follows. An extraction request is received by an extraction agent from a scientist agent (line 4). The request contains a list of queries to be evaluated by the extraction agent. The agent recursively traverses the list of queries (line 9), sending each query to the appropriate service (line 11). In our scenario, the queries correspond to requests for galaxy data, and the services correspond to astronomy data sources. The agent retrieves the result (line 17) and stores it in MySpace (line 18). Finally, the extraction agent publishes the results (line 24) and returns a URL for the published data to the scientist agent that initiated the request (line 25).

The extraction protocol expressed in MAP is clearly a straightforward implementation of the required functionality. However, there are some subtle issues in the protocol that require explanation. When exchanging messages through send and receive actions, a unification of terms against the definition `agent`(ϕ_1, ϕ_2) is performed, where ϕ_1 is matched against the agent name, and ϕ_2 is matched against the agent role. For example, the receipt of the query list in line 4 of the

```
1   %extraction{
2     method() =
3       waitfor
4        (extract($qlist) <= agent($scientist, %scientist)
5         then invoke(eloop, $qlist, $scientist)
6         then invoke())
7       timeout (invoke())
8     method(eloop, $qlist, $scientist) =
9       (($head, $tail) = extractNext($qlist)
10        then ($q, $qtype) = makeQuery($head)
11        then query($q) => agent(_, $qtype)
12        then invoke(ewait)
13        then invoke(eloop, $tail, $scientist))
14      or invoke(eend, $scientist)
15    method(ewait) =
16      waitfor
17        ((result($res) <= agent($name, $qtype)
18          then store($name, $res) => agent(_, %myspace)
19          then invoke(ewait))
20        or (noresult() <= agent($name, $qtype)
21            then invoke(ewait))
22      timeout (e)
23    method(eend, $scientist)
24      $resulturl = publishResults()
25      result($resulturl) => agent($scientist, %scientist)}
```

Fig. 6. Data Extraction Protocol

protocol will match any agent whose role is `%scientist`, and the name of this agent will be bound to the variable `$scientist`. This unification is particularly useful when we do not know the exact name of the agent in question. For example, in line 11 of the protocol we use a wild-card `_` to send the message to all agents that match the role given by `$qtype`.

The semantics of message passing corresponds to non-blocking, reliable, and buffered communication. Sending a message will succeed immediately if an agent matches the definition, and the message will be stored in a buffer on the recipient. Receiving a message involves an additional unification step. The message supplied in the definition is treated as a template to be matched against any message in the buffer. For example, in line 17 of the protocol, a message must match `result($res)`, and the variable `$res` will be bound to the content of the message if the match is successful. Sending will fail if no agent matches the supplied terms, and receiving will fail if no message matches the template.

The send and receive actions complete immediately (i.e. non-blocking) and do not delay the agent. Race conditions are avoided by wrapping all receive actions by `waitfor` loops. For example, in line 16 the agent will loop until a message is received. If this loop was not present the agent may fail to receive the reply, and the protocol would terminate prematurely. The advantage of non-blocking communication is that we can check for a number of different messages. For example, in lines 17 and 20 of the protocol, the agent waits for either a `result` message or an `noresult` message. A `waitfor` loop includes a `timeout` condition which is triggered after a certain interval has elapsed. In our example, we do not know how many agents match the `$qtype` role in line 11, so we need to wait until no further replies have been returned. This is achieved by the timeout in line 22, although we note that it would be perfectly possible to define an alternative protocol which avoids the need for the timeout.

The operations in the protocol are sequenced by the `then` operator which evaluates op_1 followed by op_2, unless op_1 involved an action which failed. The failure of actions is handled by the `or` operator. This operator is defined such that if op_1 fails, then op_2 is evaluated, otherwise op_2 is ignored. The language includes backtracking, such that the execution will backtrack to the nearest `or` operator when a failure occurs. Similarly, the body of a `waitfor` loop will be repeatedly executed upon failure, and the loop will terminate when the body succeeds (or a timeout occurs). Our language also includes a `par` operator which evaluates op_1 and op_2 in parallel. This is useful when we wish to perform more than one action simultaneously, though we do not use this in our example.

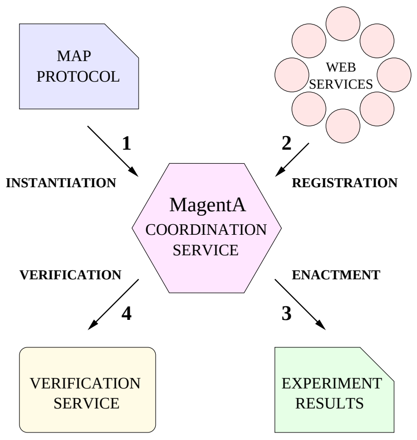
Literal data is represented by constants c in our language, which can be complex data-types, e.g. currency, flat-file data, multimedia, or XML documents. MAP does not perform any computation directly on the constants. The interpretation of the data is performed entirely by the external web services.

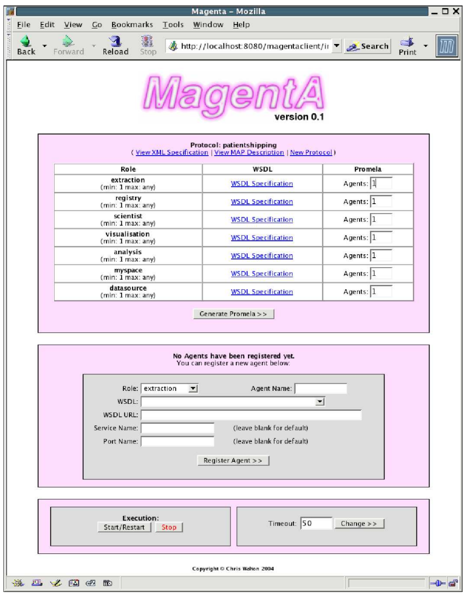
3.2 The MagentA Platform

We have implemented a platform for coordinating web services founded on our MAP language. Figure 7 illustrates the four main tasks performed by the platform. The platform closely follows the close-coupled coordination technique which we have discussed previously. At the core of the platform is the MagentA (Multi-agent Architecture) service which performs the task of the coordination service at the centre of Figure 4. The platform has been implemented in Java, and utilises the XML representation of MAP.

A screen capture of the web front-end to the MagentA service is also shown in Figure 7. The upper section of the screen displays all of the roles which are found in the protocol. There are links in this section for generating WSDL documents for each of the roles, and for verifying the protocol. The middle section of the screen is used for registering web services against the roles in the protocol. To register a service, the role, agent name, and WSDL URL are entered into the form. A port and service name within the web service can also be optionally entered. The lower section of the screen is used to initiate and terminate the enactment process. Our tool enables experiments to be rapidly constructed and configured.

The MagentA service must be instantiated with a MAP protocol before coordination can be performed. Therefore, the first task (labelled 1 on the LHS of Figure 7) in the construction of an experiment on the platform is the definition





http://localhost:8080/magentaclient/i

MagentA
version 0.1

Protocol: patientshipping
(View XML Specification | View MAP Description | New Protocol)

Role	WSDL	Promela
extraction (min: 1 max: any)	WSDL Specification	Agents: 1
registry (min: 1 max: any)	WSDL Specification	Agents: 1
scientist (min: 1 max: any)	WSDL Specification	Agents: 1
visualisation (min: 1 max: any)	WSDL Specification	Agents: 1
analysis (min: 1 max: any)	WSDL Specification	Agents: 1
myspace (min: 1 max: any)	WSDL Specification	Agents: 1
datasource (min: 1 max: any)	WSDL Specification	Agents: 1

Generate Promela >>

No Agents have been registered yet.
You can register a new agent below:

Role: extraction   Agent Name:
WSDL:
WSDL URL:
Service Name: (leave blank for default)
Port Name: (leave blank for default)

Register Agent >>

Execution:
Start/Restart Stop

Timeout: 50 Change >>

Copyright © Chris Walton 2004

Fig. 7. MagentA Platform Architecture & Web Front-End

of the protocol in MAP. The MagentA service is fully generic, in that it can perform coordination for any protocol, though only one such coordination at a time. Therefore, the instantiation process configures the MagentA service for a single protocol, which persists until enactment is complete.

A MAP specification is constructed with respect to a collection of web services that will be utilised in the experiment. In particular, each role in the protocol corresponds to one or more web services. For example, the extraction role defined in Figure 6 corresponds to a web service which supplies an implementation of the `extractNext`, `makeQuery`, and `publishResults` procedures. It is also possible to define protocols for which no actual web service exists. In this case, the MagentA service can provide an appropriate WSDL specification for a role, which can then be refined into an actual web service. Before enactment can be performed, it is necessary to register each of the web services in the experiment with the MagentA service. Registration is performed by supplying a URL to the WSDL document associated with each service, together with the role that it implements. At least one web service must be registered for each role before enactment can take place. We intend to support the discovery of services in the future, but at present the registration process is performed manually.

The registration of a web service causes an agent to be generated within the MagentA service to act as a proxy for the web service. It is important to note that the agents are generated automatically during registration. The only components that need to be supplied are the protocol, and a mapping from web services to roles. Furthermore, it is not necessary to understand the use of agents to be able to define an experiment. The third task provided by the MagentA service is the enactment of the experiment. This is where the protocol is enacted by the agents which have been defined during registration. The agents follow the protocol as a script, and invoke the external web services when necessary. The experiment terminates when all the protocol steps have been enacted, or if the protocol fails. An experiment will typically include a service that outputs the experiment data in a suitable form. The MagentA service itself does not provide any data visualisation, other than supplying a trace of the enactment.

We have noted that enactment can terminate if the protocol encounters a failure. By this, we mean either a failure internal to the protocol, or an external failure relating to web service invocation. We can avoid external failures by defining appropriate fault and timeout conditions in our protocols. Internal failures occur when a protocol backtracks to the beginning, or fails to terminate. Usually these kinds of failures will occur as a result of unexpected interactions between the agents. Ideally we would like to avoid internal failures as they can be costly, for example, if they occur during a long-lived or non-repeatable experiment. Therefore, the final task provided by MagentA is a protocol verification service, which can detect a range of protocol failures statically before enactment.

4 Conclusion

The purpose of this paper was to demonstrate that we can use existing agent technology to assist in the construction and enactment of e-Science experiments. Specifically, our previous research on agent protocols can be readily adapted to the composition of web services. Our technique is founded on the MAP language, which is a formally-defined agent coordination language. We have constructed a tool based on this language, called MagentA, which allows experiments to be rapidly constructed, verified, and enacted. Finally, we have sketched how our MAP protocols can be used to define the Brightest Cluster Galaxy experiment.

The MAP language is a lightweight formalism, providing only a minimal set of operations. This was a deliberate choice as it allowed us to define the language and the type system without unnecessary complication. However, we are now considering many enhancements to the language that would make it more suited to e-Science computation. These enhancements include: support for large datasets through an extension of the type language; support for long-lived computation, e.g. by allowing break-points in the protocols; database integration for better handling of experiment data; and support for the composition of protocols into larger experiments at the scene level.

The MAP language can be used to encode a wide range of protocols, as previously demonstrated in our work on agent protocols. However, the hand-encoding of protocols into the MAP formalism remains a time-consuming process. We are therefore currently considering a number of approaches which will permit protocols to be constructed in a more efficient manner. The simplest approach is the provision of a graphical tool for constructing protocols. Beyond this, we would like to support the automatic generation of protocols. We have made some progress into the construction of protocols as an outcome of a planning process. However, from an e-Science perspective we would ideally like protocols to be generated as a result of workflow mapping. Nonetheless, there remain significant issues in this approach, particularly as there is no real consensus on the most appropriate workflow formalism for e-Science.

A further issue that we intend to address, concerns the discovery of web services. At present, the web services that we use to define an experiment must be known in advance, and must be explicitly registered before enactment. Furthermore, the protocol must be defined to precisely match the WSDL definition of the web service. We would like to relax these restrictions, and allow a more flexible kind of coordination, which allows for (semi-)automatic web service discovery and invocation. For this, we will need semantically annotated web services, on which we can reason about the behaviour of the services. This is currently an active area of research in the Semantic Web community.

References

1. P. Allan, B. Bentley, C. Davenhall, S. Garrington, D. Giaretta, L. Harra, M. Irwin, A. Lawrence, M. Lockwood, B. Mann, R. McMahon, F. Murtagh, J. Osborne, C. Page, C. Perry, D. Pike, A. Richards, G. Rixon, J. Sherman, R. Stamper, and M. Watson. AstroGrid. Available at: `www.astrogrid.org`, April 2001.
2. D. Booth, H. Haas, F. McCabe, E. Newcomer, M. Champion, C. Ferris, and D. Orchard. *Web Services Architecture*. World-Wide-Web Consortium (W3C), August 2003. Available at: `www.w3.org/TR/ws-arch/`.
3. The OWL Services Coalition. OWL-S: Semantic Markup for Web Services. Available at: `www.daml.org/services/`, November 2004.
4. D. de Roure, N. R. Jennings, and N. Shadbolt. The Semantic Grid: A future e-Science infrastructure. In F. Berman, G. Fox, and A.J.G. Hey, editors, *Grid Computing - Making the Global Infrastructure a Reality*, pages 437–470. John Wiley and Sons Ltd., 2002.
5. M. Esteva, J. A. Rodríguez, C. Sierra, P. Garcia, and J. L. Arcos. On the Formal Specification of Electronic Institutions. In *Agent-mediated Electronic Commerce (The European AgentLink Perspective)*, number 1991 in Lecture Notes in Artificial Intelligence, pages 126–147, 2001.
6. D. Fensel and C. Bussler. The Web Service Modeling Framework (WSMF). *Electronic Commerce Research and Applications*, 1(2), 2002.
7. Layna Fischer, editor. *The Workflow Handbook 2004*. Future Strategies Inc., 2004.
8. I. Foster and C. Kesselman, editors. *The Grid 2*. Morgan Kaufmann, 2004.
9. C. Walton. Model Checking Multi-Agent Web Services. In *Proceedings of the 2004 AAAI Spring Symposium on Semantic Web Services*, Stanford, California, March 2004. AAAI.
10. C. Walton and D. Robertson. Flexible Multi-Agent Protocols. In *Proceedings of UKMAS 2002. Also published as Informatics Technical Report EDI-INF-RR-0164, University of Edinburgh*, November 2002.
11. C. Wroe, C. Goble, M. Greenwood, P. Lord, S. Miles, J. Papey, T. Payne, and L. Moreau. Automating Experiments Using Semantic Data on a BioInformatics Grid. *IEEE Intelligent Systems*, 19(1):48–55, 2004.