# An Active Blackboard for Service Discovery, Composition and Execution

**Abstract.** Organisations nowadays are in the process of developing network-enabled systems through which they deliver electronic services to citizens, customers and enterprises. Often, such services need to be combined in order to cover all aspects of a service consumer's life event. The composition of different services though is usually left to the service consumer, who needs to manually locate the individual services and drive the process of obtaining results from some services and feeding them as input to subsequent ones until all relevant services have been executed. Although it would be possible for organisations to improve their level of service through provision of composite services, realizing thus the concept of one-stop government, such facilities have not been made yet widely available. This shortage stems partly from financial considerations, since the frequent changes in the regulatory framework of both the individual services and in their interoperation requirements or in the technical aspects of the service implementation render the development and maintenance of composite services inexpedient and partly from technical issues, since format or representational incompatibilities in parameters and results hinder automation developments. In this paper we present an active blackboard architecture, which automates the task of service composition based on the semantics of individual services and the data dependencies between them. The blackboard incorporates registries, which can be employed for facilitating service discovery and an execution engine that arranges for dynamic service composition and execution.

**Keywords:** e-government; electronic services; service composition; blackboard architecture; semantics; ontologies

## Introduction

Electronic government and e-commerce systems employ information and communication technologies for offering to citizens and enterprises business services in an on-line fashion, alleviating the need for physical presence, speeding up transactions and saving resources, both for the service provider and for service consumers [European Commission,

1999); (World Bank, 2001); (Rayport, 2003)]. Both these classes of systems are highly prone to changes for a number of reasons: new services need to be provided; existing services are improved; the regulatory framework for service provision is modified, including changes to the service inputs, outputs or internal processes; non-functional parameters of services (quality, security, response time etc) are revised, triggering thus maintenance activities for the relevant information system (European Commission, 2003).

In order to address changes in their environment, software systems should be able to evolve with time and adapt to the newly emerging requirements [(Patel, N, 2003); (Hirschfeld, 2002); (Lohmann, 2004)], and provide transparent accommodation of modifications and expose a high degree of tailorability [(Theotokis, 1999); (Stamoulis, 2003); (Eardley, 2003)]. Besides the adaptability, tailorability and transparency aspects of individual information systems, systems in the context of e-government and e-business should possess some additional properties in order to deliver high quality services to their users. Firstly, prospective service users should be able to *locate* existing services in a manner that is focused on the *service consumer needs*, rather than the systemic aspects (Dogac, 2002), as is the case with existing service discovery mechanisms such as UDDI. Secondly, in many cases individual services need to be combined in order to deliver full-fledged services to the service consumer. For instance, the electronic submission of a tax return form should be combined with e-banking services to cater for payment of due taxes; the issuance of a marriage license would involve the issuance of birth certificates for each of the spouses, which in turn could involve the issuance of immigration certifications etc. In many cases, the services that are involved are not directly linked; linkage is implicit through the fact that the output produced by some service is consumed as input by another one. Besides the issue of the identification of involved services, the issue of their scheduling and execution monitoring must be addressed as well: some executions could proceed in parallel (e.g. the two birth certificates in the context of a marriage license), while other tasks include data dependencies and should thus be scheduled sequentially. No provision is also made by existing mechanisms to exploit *validity periods* of documents produced by electronic services. For example, a tax clearance certificate that is issued for a citizen at some point in time could remain valid for the next three months, thus any need for the specific document within the document validity period could be well-serviced by a copy of the already issued one, removing the need for an additional service execution. Finally, in existing approaches the *service composition paths*, i.e. the services that need to be combined to form a composite service, their data dependencies, scheduling parameters etc. have to be known a

priori and be explicitly coded in some executable specification (programming language, workflow etc). However, a mechanism that would derive the possible service composition paths that can lead to a designated result would be of great value to service consumers. This mechanism could also choose the most appropriate service composition path, if multiple ones exist; appropriateness can be perceived in terms of service availability, execution speed, current system load or other relevant aspects.

In this work we present an architecture that employs the blackboard paradigm [(Hayes-Roth, 1985); (Nii, 1986); (Corkill et al., 1987); (Engelmore and Morgan, 1988); (Jagannathan et al., 1989); (Corkill, 1991); (Carver, 1997)] to address the issues identified above. The proposed blackboard is an *active one*, i.e. it does not constitute a simple repository of information available to the authorised entities; rather, it encompasses mechanisms for deducing service composition paths, driving the process of composite service execution, maintaining a cache of documents and performing transformations where appropriate. The blackboard employs an ontology for presenting a semantically rich view of the service pool to prospective service consumers.

The rest of the paper is organised as follows: The next section presents related work and background information for blackboard architectures and service composition. Afterwards, the proposed architecture is outlined, its components are identified and their functionality is discussed and exemplified through real-world scenarios. Finally, conclusions are drawn, the main contributions of the proposed solution are recapitulated and future work directions are drawn.

## Background and Related Work

Blackboard systems have been used for more than two decades (Erman, 1981) as a highly structured, special case of opportunistic problem solving (Nii, 1986). Within a blackboard model, domain knowledge regarding the inputs, outputs as well as intermediate and partial solutions needed in the context of problem solving need to be well-organised, so as to be exploited by knowledge modules accessing the blackboard to derive the problem solution. The blackboard paradigm for problem solving supports the incremental problem solving (Carver and Lesser, 1994) in the sense that the process starts off with only the input data available, then some knowledge module derives an additional piece of information from it and makes this piece available within the blackboard space; the newly produced information can be used (together with the initial input data) by all other knowledge modules that will in turn produce additional

information, which will be again incorporated in the blackboard space. Thus, various independently working modules contribute jointly to the problem solution. The blackboard paradigm has also been used for real-life applications such as the PLAN component of the Mission Control System for RADARSAT-1 (Canadian Space Agency, 2005) described in (Corkill, 1997)

Service composition, on the other hand, has insofar proceeded by using pre-determined execution scenarios in order to model composite services. One such example is the eGov project (Tambouris, 2001), in the context of which simple services can be combined to form composite, life-event oriented electronic services. The synthetic task is undertaken by the composite service developer, who must first discover the simple services, import them to the development environment and finally arrange the flow of execution and the linkage of inputs and outputs. In this paradigm, if the inputs, the invocation method or the outputs of any of the simple services change, the composite service developer should accordingly amend the composite service model. Similar approaches are taken by technological frameworks, such as (Bunting et al, 2003).

The rigidity in the execution path specification is relaxed in the model described in (Casati, 2000), where the composite service developers prescribe some composite service schema in terms of flow structure, definition of service, decision and event nodes, data processing, and transactional regions. The prescribed schema may be modified in run-time (subject to authorisation constraints), to cater for variations to any of the composite service schema elements. This work additionally introduces consistency rules, which are able to verify that changes that have occurred either in the composite service schema or in some constituent service have rendered the composite service to be invalid; however still human intervention is required to remedy these cases.

In (Lepouras et al., 2005), a first version of the active blackboard architecture was proposed as a means for web services composition. In this paper we extend this first approach by introducing rich vs. simple replies, additional semantics – most important of them being the *updates* and *if-updated*, that correspond to event-condition-action rules – and a cache manager scheme has been introduced to the system. Also issues that arose during the implementation have been resolved and are discussed in the implementation considerations section.

In (Peristeras, 2006) the authors present a service metadata representation and service composition model – namely GEA Service model – which shares functionality with the model presented in this work and in (Lepouras, 2005); however this model does not clearly distinguish between generic service provisions and concrete service implementations; this feature is necessary for avoiding duplication of information and

inconsistencies among service implementation. Moreover, advanced features such as rich vs simple replies and caching of intermediate results are not considered. The GEA Service model has been implemented using semantic technologies within the SemanticGov project [(Loutas, 2007)]

## The Active Blackboard Architecture

This section describes the proposed approach. Firstly, the overall architecture is described in terms of components, their functionality and interoperation. Subsequently, for each component, a detailed description is given and its key aspects are discussed; implementation considerations for the different modules and the data space are also outlined. Real-world examples are given in the text to exemplify the use of the platform.

### Overview of the Active Blackboard Architecture

The active blackboard architecture encompasses functionality both for electronic service providers and consumers. Electronic service providers are able to *register their services* with the blackboard, making them thus available for invocation to service consumers, either directly indirectly. A service is invoked directly if a consumer asks explicitly for it; indirect execution occurs when a service consumer requests service A and this needs an input produced by service B. In such a case the service execution engine arranges so that service B is executed to produce the desired result; this arrangement is transparent to the service consumer, with the possible exception of further input required by service B (discussed in detail later in this section). The process of service provision and consumption is semantically enriched through an ontology, which covers both services and data managed by them. In order to further enhance service interoperability, the active blackboard system includes the provision for executing methods that transform instances of ontology nodes to instances of other ontology nodes. Finally, the active blackboard is able to keep copies of service parameters and results within a local cache, enabling therefore the reuse of results which are still valid, in order to minimise processing and communication and optimise the overall turnaround time. The overall architecture of the active blackboard is illustrated in Figure 1. In the following paragraphs, the modules of the active blackboard are described in detail.

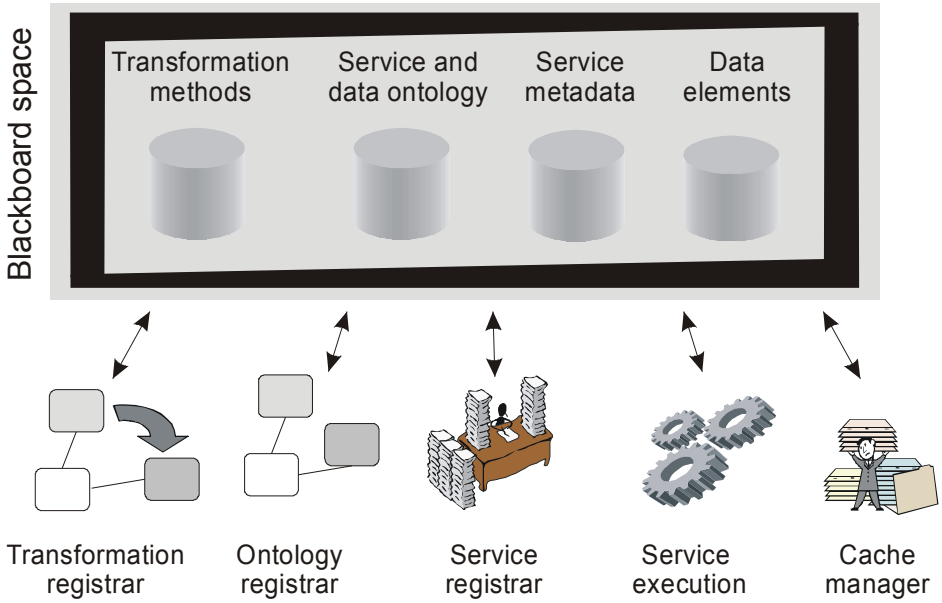


**Figure 1 – The active blackboard architecture**

### The ontology registrar

The ontology registrar manages the *service and data ontology*, which is the semantic infrastructure for service classification, discovery and interoperability. The ontology models the basic concepts related to services, which are:

1. the organisations that provide electronic services
2. data passed as parameters to services and their structural and semantic descriptions
3. services offered by organisations and their semantic descriptions

Figure 2 depicts part of the *service and data ontology*, as modelled using the KAON tool (KAON, 2004). The rectangle nodes are *concepts*, which are used to model classes of objects, while the rounded rectangle nodes are *instances of concepts,* which map to individual objects. For clarity reasons the figure only shows the *is-a* relationships between concepts and between concepts instances (e.g. Ministry of Finance *is-a* Ministry *is-a* organisation); more relationships between concepts and instances are generally established, the most important of which being the *offered-by, managed-by, consists-of, produces, is-related-to, updates* and *if-updated* relationship types. The *offered-by* relationship links a *service instance* with an organisation concept or instance, and dictates which organisation or *class of organisations* has the authority to offer the specific kind of service. For example, the "Birth Certificate" service instance node has two

emerging links of type *offered-by* connecting it to the "Municipality" concept and the "Ministry of Internal Affairs" concept instance (any instance of the municipality concept can issue birth certificates and the Ministry of Internal Affairs can also issue birth certificates). The *managed-by* link type connects an organisation to (a) instances of the service concept (b) instances or sub-concepts of the "Data" concept and (c) instances or sub-concepts of the "Organisation" concept. Such a link indicates which organisation is administratively responsible for the service or datum and can thus define further structural properties of the related node. For example, the Ministry of Internal Affairs is administratively responsible for the "National Id No" instance and can thus define that the valid format of a national identity number is one or two characters followed by six digits. Note that the administratively responsible organisation is also responsible for establishing the *offered-by* links.

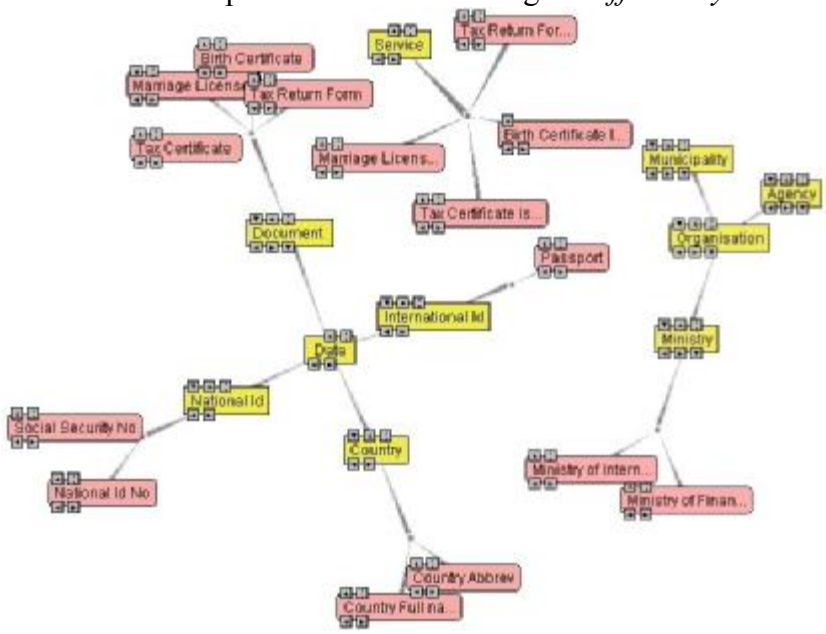


**Figure 2 – Part of the *service and data ontology***

The *consists-of* relationship type provides the ability to model cases in which a data type is composed of a number of smaller data elements. The *consists-of* relationship type has a single source and multiple targets, indicating that any instance of the source node has *all* its target nodes as elements. For example, a *consists-of* link emanating from the "International Id" node and pointing to the nodes "National Id" and "Country" states that any international identity can be modelled (in addition to the Passport, which represents an international identity by virtue of the *is-a* relationship) through a pair of a "National Id" and a

"Country". Note that a "National Id" may be represented either as a "National Id No" or a "Social Security No".

The *produces* relationship type links a single service to the document it produces. The node to which the link arrives must be a descendant of the "Document" sub-concept of the "Data" concept. For example, the "Marriage License Issuance" service is linked with a *produces*-type link to the "Marriage License" document.

The *is-related-to* relationship type emanates from a single sub-concept or instance of the "Data" concept and arrives to one or more descendants of the "Data" concept. The purpose of this relationship type is to link *the result of a service* (or a *data element*) with a *set of data values* that uniquely identify this result (or the data element) modelling thus *functional dependencies* (Halpin, 2001). For example a "Personal Address" descendant of the "Data" concept is linked to the "National Id" node to signify that a specific person always has a specific address; a "Birth Certificate" document (result of the "Birth Certificate Issuance" service) is also linked via an *is-related-to* link to the "National Id" node to designate that the birth certificate pertains to the citizen whose identity is used in the "Birth Certificate Issuance" service invocation. Similarly, a "Car Insurance" document can be linked to a "National Id" and a "Car Licence Plate No" to designate that the car insurance document is associated with *the specific combination* of driver and car. It is possible for a *is-related-to* relationship to point to a specific descendants of the "Data" concept more than one times: for example, a "Marriage Licence" document points to the *National Id* concept *twice* (and once to a *timestamp* concept), since it is uniquely identified by the two persons that will be married (and the timestamp).

The *updates* relationship emerges from a single descendant of the "Service" concept and point to a single descendant of the "Data" concept, indicating that the service updates the information represented by the specified data construct. For example, an "Announcement of Move" service is linked to the "Address" node to indicate that the service modifies an address. The entity whose address is modified can be traced using the *is-related-to* relationships emerging from the datum, if any. Conversely, the *if-updated* relationship starts from a single descendant of the "Data" concept and points to a single descendant of a service to signify that when a specific datum is updated, the designated service should be invoked. Effectively, the *updates* and *if-updated* relationships model data-triggered service executions, removing the need for a service to explicitly call other services that need to be executed in specific situations. All appropriate service executions are automatically scheduled by the service execution module.

Each node in the ontology has a number of *attributes* that describe various aspects of it. An instance of a *data* sub-concept has attributes dictating the valid format(s) for this element, a list of allowable values (e.g. the "Day of Week" node will list the days from Sunday to Saturday), minimum and maximum bounds, the confidentiality level of the datum (e.g. publicly known vs. strictly personal), multilingual labels for its description etc. Attributes for organisations include descriptive data (e.g. physical location, phone and fax numbers) and *identification credentials* (user names and passwords, IP addresses, SSL keys etc) which are used by the registrar services to validate that connecting entities as acting on behalf of the specific organisation. Finally, attributes for sub-concepts and instances of the "Service" concept include the confidentiality level, a *time to live* – i.e. for how long after its issuance by the respective service its result remains valid and an indication whether the service result is *reusable*, that is whether results of the specific service may be retained and used as input in a subsequent service invocation. The latter two properties are exploited by the cache manager upon service execution.

The ontology can be modified and queried through the *ontology registrar* module. Organisations wishing to modify portions of the ontology that they are responsible for (as indicated by the *is-managed-by* relationships) submit the relevant requests to the *ontology registrar*. Each request is complemented by appropriate credentials to identify the submitting entity as acting on behalf of the organisation. The service registrar validates the credentials and verifies that the affected ontology portion is actually managed by the requesting organisation before honouring the request. The ontology may be queried by prospective service consumers to determine whether a specific service is listed in the ontology, as a first step towards service invocation (further steps are described in the following paragraphs). Additionally, governmental or third party portals may query the ontology to extract services listed in the ontology, in order to present their users with lists of available services. Note that such a service list may be simply alphabetically sorted, categorised according to the responsible (or issuing) organisations, or matched against some external classification scheme, e.g. a *life-event* ontology maintained by the portal. A further option for portals is to query the available *documents* in the ontology and display them to their users. When the portal user selects the desired document, the portal may reversely traverse the *produces* links to locate the service(s) that produce the selected document.

**The service registrar**

The service registrar manages the *service metadata* repository, which links the *semantic representations* of services within the ontology with *concrete implementations* that actually deliver a service. Organisations that offer a service should *register* the service to the service registrar providing the following information:

- a link of type *implements* to a descendent of the "Service" concept in the ontology, which designates which abstract service this concrete implementation realises.
- zero or more links of type *uses*; each link points to a single descendant of the "Data" concept. These links actually model the *input data* that the service needs for its execution, which may be either values that should be entered by users or documents that will be produced as the result of execution of other services. In such cases, it is mandatory for the links to be *labelled* using a *role name*, in order to allow the system to distinguish the inputs. For example, the "Marriage License" service includes two links of type *uses* to the "Birth Certificate" document indicating that two birth certificates should be provided as input to the marriage license service (one certificate per spouse); one of these links will be tagged with the role name "husband", while the other link will be tagged with the role name "wife".
- a *condition*, which must evaluate to "true" to make the service eligible for delivering the abstract-level service of the ontology. Consider for example the birth certificate service: the Ministry of Internal Affairs may issue a birth certificate for any citizen of the country, since it stores information for all citizens in its database. A municipality, on the other hand, has no such ability since its database only stores information regarding its residents. In this case, the Ministry of Internal Affairs will register its service implementation with a condition equal to "true" (the implementation is always eligible to deliver the service), while the municipality of Paris will register its implementation with a condition *Address.City = "PARIS"*. The *Address* appearing in the left hand side of the condition is a descendant of the "Data" concept, to which a *uses* link is specified.
- a specification of how the service can be called. The actual calling method depends on the service implementation: for services implemented as web services (Newcomer, 2002), a WSDL specification can be provided; for services delivered through RMI (Sun Microsystems, 1999) the name of the remote object, the interface name and method should be listed etc.

Optionally, the service registration may be complemented with additional information regarding the service operation, such as cipher suit specification for encryption, authentication credentials that must be presented upon service invocation, hours/periods of service operation etc. The service registrar also allows the specification of non-functional metadata, such as anticipated turnaround time for service requests, service uptime, communication line speeds, whether the service is *idempotent* (i.e. it can be executed multiple times without any side-effect) and so on. These metadata can be exploited by the service execution engine to formulate an optimal execution plan for delivering the service to the citizen, in the presence of multiple service providers.
Note that the arrangement presented above allows for distinct concrete implementations to use different input data for producing the same result. For example, one implementation of the "Enrolment in Higher Education" service offered by the Ministry of Education may accept as input a "Higher Education Examinee Number" and perform all actions based on the examination data in the ministry's database, while some other implementation offered by a University may require the student's identity number, name and surname and the department to which the student will enrol. Both service implementations produce an "Enrolment Certificate" document as output.
Similarly to the ontology registrar, the service registrar verifies that appropriate credentials are provided within the request to identify the submitting entity as acting on behalf of a specific organisation and that this organisation is allowed to deliver the service.

**The transformation registrar**

The transformation registrar manages the *transformation method* repository, whose entries specify how certain descendants of the "Data" concept in the ontology can be transformed to some other descendants of the "Data" concept. These transformations generally include database lookups and filling-in of default values, and are provided for the convenience of the end users of services. Consider for example the case that some service of the Ministry of Health requires the social security number of the citizen, for identification purposes, however the citizen only has the driver license at hand. In the presence of a suitable transformation method, the service user could enter the driver license number and this would be mapped to the social security number, thus the service would proceed. Similarly, the citizen's identity number and country could be derived from the passport number, while an Austrian VAT number can be transformed to a European VAT number by prepending the constant string

"AT". Such transformations can be automatically performed by the service execution module.
For a transformation to be registered to the blackboard, the following information should be provided to the transformation registrar:

- one or more links of type *uses* to descendants of the "Data" concept
- one or more links of type *computes* to descendants of the "Data" concept
- a specification of how the transformation will be performed.

The semantics of such a registration request is that *all* the data pointed to by the *uses* links will be used to compute *all the data* pointed to by the *computes* links, using the designated method. For example, the transformation from a driver license number to a social security number will be registered using a single *uses* link, pointing to the "Driver Licence Number" instance within the ontology and a single *computes* link, pointing to the "National Id No" instance within the ontology; the transformation from a passport number to identity number *and* country, on the other hand, will be registered using a single *uses* link (to the "Passport" instance) and two *computes* links (to the "Country Abbreviation" and "National Id No" instances). Similarly to the service registrar, the actual implementation of the transformation method may be provided as a WSDL specification, RMI parameters etc. For simple transformations (such as providing default values, extracting portions of strings etc), the transformation registrar allows the specification of the transformation via a simplistic "language", that allows such operations to be performed. For example, the aforementioned transformation from an Austrian VAT number to a European VAT number can be specified as:

```
concatenate("AT ", Austrian_VAT_Number)
```

(The Austrian_VAT_Number is the ontology instance pointed to by the *uses* link.)
This provision allows for minimising the overall service execution time, since processing such a transformation rule is far more effective than invoking an external to the blackboard service.
Again, the request must be complemented with appropriate credentials to verify that the requestor is acting on behalf of a valid organisation.

**The service execution module**

The service execution module is the entry point provided by the blackboard for service consumers. Having obtained a valid service name for the blackboard by querying the ontology registrar, a service consumer may submit a request to the service execution module asking for this

service to be executed. Such a request should be complemented with any appropriate parameters that are required by the service. The blackboard will undertake the task of invoking the designated service (and, possibly, any other services that produce intermediate results or transformations to perform data coercions), and will return the result to the service consumer; under this execution scheme the service consumer interacts only with the blackboard and needs not be aware of the actual steps taken for the request to be honoured. One issue that consumers may face, however, is how to determine the parameters that should be provided to the service for the desired result to be obtained. Before listing the options available to clients, let us discuss which data sets can be used for a particular service S to be successfully executed in the context of the blackboard.

Recall that a single service within the ontology can have multiple *concrete implementations*, as registered to the service registrar. Each of these concrete implementations can define its own input parameters via the *uses*-type links to descendents of the "Data" node in the ontology. If an input parameter is a descendent of the "Document" sub-concept, it should be produced as the result of some other service execution, whereas other input parameters (non-descendents of the "Document" sub-concept) are directly provided by the service consumer.

The simplest case of service invocation is when a single concrete implementation exists and none of its input parameters is a descendent of the "Document" sub-concept. In this case, the service consumer needs to provide values for all input parameters of the unique concrete implementation of the service. For the cases that a *uses* link points towards a *sub-concept* and not an *instance*, the service consumer may provide *any* of the instances which are descendents of the sub-concept (e.g. if the *Birth Certificate Issuance* service requires a *National Id* datum as input, then either a *National Id No* or a *Social Security Number* value may be provided). Finally, if the ontology node corresponding to an input parameter is the source of a *consists-of* link, then the service consumer should provide values for *all* the targets of this link.

The transformations registered to the blackboard can be exploited in the context of a service invocation as follows: if some input parameter set includes data of types $A_1$, $A_2$, …, $A_n$ which can be computed via transformations from data of types $B_1$, $B_2$, …, $B_m$, then the inputs to the transformations rather than the data directly required by the service can be provided as inputs, and the service execution module will arrange for the proper transformations to be applied, prior to the service invocation. For example, if the tax certificate issuance requires as input a VAT number and a date and there exists a transformation that computes the VAT number through the Social Security Number, then the service may be invoked providing a Social Security Number and a date as input

parameters; the service execution module will use the registered transformation to convert the Social Security Number to a VAT number, which will be finally passed as an input parameter to the service, along with the value for the date that the service consumer has originally provided.

Figures 3 and 4 illustrate XML documents that correspond to service invocations. Note the usage of the "tag" attribute in figure 4, which is used to distinguish the role of each of the two parameters of the same type. The value of the "tag" attribute should match the tag on the "uses" link specified in the service registration.

```
<requestedService>Birth_Certificate_Issuance</requestedService>
<inputs>
	<input type="National_Id_No">
		<value>X456789</value>
	</input>
</inputs>
```

**Figure 3 – Invocation of the "Birth Certificate Issuance" service**

```
<requestedService>Marriage_Certificate_Issuance</requestedService>
<inputs>
	<input type="National_Id_No" tag="husband">
		<value>X456789</value>
	</input>
	<input type="National_Id_No" tag="wife">
		<value>Y567890</value>
	</input>
</inputs>
```

**Figure 4 – Invocation of the "Marriage Certificate" service**

In the presence of *multiple implementations* of a service, the service consumer may provide input parameters that are adequate for *any* concrete implementation. Upon reception of a service execution request, the service execution module will determine the concrete implementation that can be invoked using the provided values, taking additionally into account the *condition* associated with each concrete service implementation. If the supplied input parameter values render some condition(s) to evaluate to *false*, then the associated concrete implementation(s) will not be further considered by the execution engine in the context of honouring the specific request.

A more complex case is when some input(s) of the requested service S is a descendent of the "Document" concept in the ontology. For each such input, a service S' needs to be invoked to produce the document, which will be subsequently used as input to the service requested by the service consumer. For the invocation of S' to be possible, the request submitted by the consumer should provide not only the inputs required by the directly invoked service S (except for the inputs which are descendents of the "Document" concept), but also all inputs that are needed by service S'.

Thus, if the services listed in Figure 5 exist, an invocation of the "Passport Issuance" service should provide a value for the "National Identity Card Number", which is directly needed by the invoked service, plus a value for "Address", which is needed by the "Residence Certificate Issuance" service that will produce the "Residence Certificate" required by the "Passport Issuance" service. Note that the value for "National Identity Card Number" needs to be provided only once, and the execution module will arrange so that the value will be passed to both service executions. Such a chain of executions may extend to any number of levels. Transformations may again be employed for deriving the values of input parameters, as described for simple service invocations.

| *Service* | *Inputs* | *Output* |
|---|---|---|
| Passport Issuance | National Identity Card Number<br>Residence certificate | Passport |
| Residence Certificate Issuance | National Identity Card Number<br>Address | Residence certificate |

**Figure 5 – Services that must be sequentially invoked**

If a service consumer does not know which parameters are needed for invoking a specific service, a relevant *query* may be submitted to the service registrar module. In such a query the desired *service name* will be stated and a flag can be providing indicating that a *rich reply* is requested. A rich reply will list all parameter combinations that can be supplied to the service, while a *simple reply* will only provide one parameter set including the types directly pointed to by the relevant *uses* links. If multiple implementations for a service are provided with distinct parameter sets, a simple query will return the parameter set for an implementation without an associated *condition*; if all implementations have an associated condition, one parameter set will be chosen arbitrarily. The option for a simple reply is given to facilitate the work of service consumers that do not include extensive parsers or logic to handle rich replies. Figure 6 presents an example for a rich reply to a service parameter query, whereas Figure 7 presents an example for a simple reply.

```
<service>Birth_Certificate_Issuance</service>
<inputParams>
    <option>
         <param type="Social_Security_No" base_type="xs:integer"/>
         <param type="National_Id_No" base_type="xs:string"/>
         <param type="Passport_No" base_type="xs:string"/>
    </option>
</inputParams>
```

**Figure 6 – Rich reply for parameters to the Birth Certificate Issuance Service**

```
<service>Birth_Certificate_Issuance</service>
<inputParams>
      <param type="Social_Security_No" base_type="xs:integer"/>
</inputParams>
```

**Figure 7 – Simple reply for parameters to the Birth Certificate Issuance Service**

Note that both parameter queries and service invocations presented in the figures above make use of the XML syntax, and service consumers need to be aware of this protocol to interact with the blackboard. Since numerous service consumers exist that directly support the web service paradigm only, the blackboard arranges so that for each available service an appropriate WSDL file is created and a mini-stub is generated to accept invocations for the service according to the web service paradigm. All available services are also registered with a UDDI repository, which allows service consumers that are not aware of the blackboard communication protocol but are compliant to the standard WS specifications, to locate and subsequently invoke the services offered through the blackboard.

Event-condition-action triples, such as the paradigm presented in (Papamarkos, 2003), can be an alternative formal model for the execution module of the blackboard architecture. Event-condition-action modelling allows for direct use of existing validation engines, but lacks the high-level semantics offered by ontology modelling. In the paradigm presented in (Papamarkos, 2003), the event-condition-action rule for creating a document of type *dtype* can be expressed as follows:

ON REQUEST (dtype, parameter_set)
PREPARE REQUEST(uses1, parameter_set), REQUEST(uses2, parameter_set), …
IF $uses_i$*_condition*
DO EXECUTE SERVICE (concrete_implementation, uses1, uses2, …)

The event in this rule is the submission of a request for a document of type *dtype*, providing a set of parameters *parameter_set*, which may be either documents or date values. The IF clause is the condition part of the rule and corresponds to the guard condition associated with the concrete implementation, which is finally invoked as specified in the DO clause (*action* part of the rule). The PREPARE clause is a *preparatory action* specification, modelling the use of the parameters provided by the service consumer to produce the documents and/or values actually needed by concrete implementation to be invoked: for example, if the provided parameter_set contains a social security number, but the concrete implementation requires a VAT number, then the relevant PREPARE clause will be

PREPARE REQUEST(VAT_number, {SSN})

to compute the VAT_number needed for the concrete service invocation. Notice that the preparatory step is itself a REQUEST event, firing thus

additional ON REQUEST rules. Instead of an EXECUTE SERVICE specification, the DO clause may contain an EXECUTE TRANSFORMATION specification, which is functionally equivalent but instead of using a service to do the computation, it employs an appropriate transformation.
Two more forms of event-condition-action rule are needed to model the operation of the execution module. These forms are:
ON EXECUTION (service, parameters)
DO UPDATE (datum1, parameters), UPDATE(datum2, parameters), …
and
ON UPDATE (datum, parameters)
DO EXECUTE SERVICE (service1, parameters), EXECUTE SERVICE (datum2, parameters), …
The first rule form actually models the "updates" relationship of the ontology, stating that the designated service updates the data specified in the DO clause. The update method need not be listed here, since it is already coded in the service implementation; this clause actually registers an UPDATE event for the datum in the event-condition-action environment, which triggers the firing of the respective ON UPDATE rules, modelling the "if-updated" relationship of the ontology. When such a rule is fired, the services listed in its DO clause are executed, fully thus supporting the data-driven service executions presented above.

**The cache manager**

In general, documents and certificates produced by public authorities have a *period of validity*, within which they may be used in transactions. For example, a tax clearance certificate, attesting that the citizen has no due taxation debts, may be valid for three months after being issued and within this time period it may be used in the context of any transaction with the government. The blackboard exploits this property of the documents, in order to re-use documents issued by services, avoiding thus service re-execution and obtaining benefits both in terms of reducing overall system load and minimizing total service delivery time.
In order to achieve these goals, the cache manager cooperates with the service execution module as follows:

1. When the service execution module invokes some service that produces a document, it examines the service and data ontology metadata to determine whether the document is cacheable, i.e. whether it can be stored and re-used. If the document is cacheable, the *is-related-to* links emanating from the document node within the ontology is traversed to determine the data elements that have been used as parameters to the service and uniquely identify the document

(e.g. a *birth certificate* will be linked through an *is-related-to* link to the National_Id_No ontology node). The document along with the values of the parameters, are presented to the cache manager for storage.

2. The cache manager accepts the document and the related parameters and extracts from the service and data ontology the *time to live* for the specific document type. The cache manager stores in its database the parameters, the document and the *document expiration time*, computed by adding to the current instant the *time to live* extracted by the ontology.
3. When a service needs to be invoked to produce a document, the service execution module first checks whether such a document exists in the cache by presenting the identifying parameter values and the document type to the cache manager. If a document of the given type and associated with the specific parameter values exists in the cache, then the cache manager verifies that the document remains valid by comparing the *document expiration time* with the current time instant. If the document is valid, it is returned to the service execution module, which will then use it directly, instead of invoking the service that would produce the document. If however such a document does not exist or has expired, service invocation is actually performed.

In order to optimise cache usage, the cache manager periodically scrutinises the cache to locate documents that have expired and removes them from the cache. Such a purging procedure will free up cache space for new documents to be stored. If the space allocated to the cache fills up, then the cache manager employs a *cache replacement algorithm* to free up space for new documents; standard cache replacement algorithms [(Wessels and Claffy, 1997); (Wang, 1999), (Zhu, 2001), (Vassilakis, 2002); (Wessels, 2004)] may be used for this purpose, but the *document expiration time* should be taken additionally into account (e.g. documents expiring in the next few days may be considered as prime candidates to be removed from the cache).

**Implementation considerations for the blackboard architecture**

The overall blackboard architecture encompasses a number of distinct modules that access a shared information space, hosting all the information needed by the modules (service and data ontology, service metadata, transformation methods, and data elements). Since the modules are operating concurrently, access to the shared information space should be synchronized, in order to avoid cases that some module reads data that has been partially updated by some other module (note that in the

proposed architecture, each type of information within the blackboard is updated by a single module, thus lost update concurrency hazards (Date, 1994) are not bound to occur). It is also desirable for the shared information space to be able to accept *registrations to events* from modules and automatically produce *notifications* when such events take place. This would enable modules to promptly react to certain events. For example, if the service execution module is using an item from the cache and the cache manager determines that the item has expired and removes it, the removal of the item from the cache space could result to a notification for the service execution module to cease using the expired item. Taking these into account, JavaSpaces (Freeman, 1999) has been chosen for the implementation of the shared information space, since it provides space operations, distributed events, leases and transactions.
Within the execution of any single blackboard module, certain tasks may need to be performed concurrently. This is particularly true for the following cases:

1. the service execution module, which undertakes the task of accepting requests and performing the activities required for fulfilling them. Since each individual request may require considerable time to be completed, it would be unacceptable to queue requests and process them sequentially. The service execution module may also employ concurrency in the context of a single request (a) in the case that two concrete service invocations have no interdependency (i.e. neither of them requires directly or indirectly the result of the other to proceed) and (b) in the case that alternative paths for the computation of some result exist and an optimal turnaround time is called for, all paths may be executed concurrently and the first result obtained can be used for subsequent operations.
2. the cache management module may service requests for storage and retrieval of documents in a synchronous manner, while a concurrent task may sweep through the cache to locate documents that have expired and remove them from the cache. This task may also remove rarely used documents from the cache, to guarantee that cache storage space is always available for new documents.

In both these cases, the multi-thread execution paradigm (Butenhof, 1997) can be adopted to provide the required parallelism. The service execution module may spawn a new thread to service each incoming request, allowing thus activities pertaining to different requests to be executed concurrently. Alternative paths may also be handled through the multi-thread execution paradigm, with a separate thread being created to execute each path. When the first thread signals (through an appropriate

semaphore) that the result has been computed, the main thread for the task collects the result and terminates the threads computing the remaining paths, in order to save system resources. Similarly, the cache manager may create a low-priority thread to handle identification and removal of expired or rarely used documents, while requests for storage and retrieval of documents may be handled by either a single high-priority thread or by multiple, high-priority threads. Threads may be finally used by the execution module, when services need to be invoked as a result of the presence of "if-updated" type relationships; tasks that are specified to be carried out when some datum is updated may be assigned to separate threads, which will proceed asynchronously to any other task. In environments with high workload, the registrars may also employ multiple threads (create a thread for each incoming request) to provide better turnaround time. Note, also that multithreading reduces the possibility of bottleneck in the execution module.

If no multithreading is available, then in order to avoid bottlenecks at the execution module the following solution may be opted. The blackboard decides on the execution plan, and then creates a jar file containing the execution plan and sends it to the client to be executed. This is also a solution for cases where the client has been authenticated to use a certain service (e.g. paying through an e-banking system) and the blackboard would be considered as an alien, possible dangerous third party trying to impersonate the client, thus rejecting it. In any case, it should be stressed out that the blackboard only composes and coordinates the execution; it does not perform the actual execution of the services.

We haven't opted for a distributed implementation of the service composition and execution, as this way a number of issues, such as concurrency, recovery, etc, would be more complicated, with no apparent performance gain, as we just explained.

One more implementation issue has to do with the detection and resolution of circles that may exist in the service/data ontology. Under normal circumstances no circles should exist, as only administrative procedures are recorded in the blackboard, and for all citizens' sake circles should not appear in such a procedure. Note also, that procedures in the ontology are not defined algorithmically but using a data-driven approach, so the probability that the administrators should erroneously create a circle is minimised. In any case, the simple and working approach the UNIX system uses to detect and resolve circles when dealing with symbolic links may be also used in our implementation: a maximal path length is set. So, the administrators of the blackboard system may set a maximum allowed length of the service call sequence, and if a composition exceeds this length – while it is being formed – the request will fail and the

administrators will be informed in order to check the definition of the services that have taken part in the composition.
A final implementation consideration is the provision of resilience against software or hardware failures, especially in the presence of long-lasting tasks. If the software implementing a module of the blackboard architecture (especially the service execution module) or the computer hosting it crashes, it would be desirable to be able to resume tasks from the time point of the crash, rather than restarting requests (or housekeeping activities, such as the cache cleanup) from the beginning. To this end, a checkpointing mechanism should be used with the blackboard, to write to persistent store the current status outstanding transactions. Checkpoints may be taken periodically, or can be incrementally built by writing to persistent store the operations that have been completed, similarly to recovery logs in database systems (Date, 1994).

## Conclusions

In this paper we have presented a blackboard-oriented architecture, which enables service discovery, dynamic composition and execution. The architecture incorporates high-level service semantics, through the use of an ontology, and employs data dependencies both for automatically deducing the services that need to be involved in a composite service and for effectively scheduling their execution, exploiting the inherent parallelism. Metadata within the ontology can be exploited for document caching, in order to avoid redundant service execution. Finally, the blackboard can simulate the invocation paradigm of web services and RMI, enabling thus its usage through existing clients, without the need for any modifications.
An important advantage of the proposed approach is that service interfaces and implementations can change at any point in time, without the need for modifying other services that are related to them, either as input providers or as result consumers. The blackboard, acting as a mediator in such cases, will arrange for combining the new service definitions with appropriate input providers or result consumers by employing transformations, intercalating other services or modifying the inputs that must be presented to the execution process, in order to execute the composite service. No predetermined execution paths need to be defined or changed, and constituent services need not be aware of the source of their inputs or the destination of their outputs.
In the presence of multiple providers for the same simple service, the blackboard may attempt to optimise the execution plan according to a number of criteria which may include expected response time, current

system load, cost of using the specific service provider and so forth. The algorithms that may be employed for optimisation will be investigated in the context of our future work. Optimisation may also extend to the user interface level, taking into account user group diversities and specific capabilities within each group. For instance, a social worker may work better by identifying a beneficiary through the social security number, whereas citizens may prefer to identify themselves through their name, surname and year of birth. Such a provision will provide adaptability to end-user needs, complementing the adaptability to regulatory frameworks and implementation parameters. Finally, future work will address the issue of blackboard replication as a means for providing resilience against software and hardware failures and network partitioning, as well as for enhancing performance. The issues associated with replication, such as repository synchronisation, request execution handover and sibling cache exploitation will be also studied.

## References


European Commission, *Public Sector Information: A Key Resource for Europe - Green Paper on public sector information in the information society* COM(98) 585 January 1999, available from ftp://ftp.cordis.lu/pub/econtent/docs/gp_en.pdf, (last visited November 21st 2007)

World Bank, *A Definition of e-Government*, http://www1.worldbank.org/publicsector/egov/ (last visited November 21st 2007), 1999.

Rayport, F., Jaworski, B., Rayport, J., *Introduction to e-Commerce*. McGraw-Hill/Irwin; 2nd edition, April 18, 2003.

European Commission, *The Role of e-Government for Europe's Future,* Brussels, 26.9.2003, COM(2003)567 final, available at http://europa.eu.int/information_society/eeurope/2005/doc/all_about/egov_communication_en.pdf (last visited November 21st 2007), 2003

Patel, N. (editor), *Adaptive Evolutionary Information Systems*, Idea Group Publishing, 2003

Hirschfeld, R., Matthias, W., Gybels, K., "Assisting System Evolution: A Smalltalk Retrospective", *Proceedings of the ECOOP 2002 First International Workshop on Unanticipated Software Evolution (USE)*, Malaga, Spain, June 10-14, 2002.

Lohmann, D., Gilani, W., Spinczyk, O., "On Adaptable Aspect-Oriented Operating Systems", *Proceedings of the ECOOP Workshop on Programming Languages and Operating Systems (ECOOP 2004)*, Oslo, Norway, 2004

Theotokis, D., Kapos, G. D., Vassilakis, C., Sotiropoulou, A., Gyftodimos, G., "Distributed information systems tailorability: A component approach", *Proceedings of the IEEE Workshop on Future Trends on Distributed Computing*, Cape Town, pp. 95-101, 1999.
Stamoulis D et al, "Ateleological Development of 'Design-Decisions-Independent' Information Systems". *N. Patel (ed) Adaptive Evolutionary Information Systems*, Idea Group Publishing, 1999.
Eardley A. et al, "Methods for Developing Flexible Strategic Information Systems: Is the Answer Already Out There?" *N. Patel (ed) Adaptive Evolutionary Information Systems*, Idea Group Publishing, 1999.
Dogac, A., Cingil, I., Laleci, G., Kabak, Y., "Improving the Functionality of UDDI Registries through Web Service Semantics", *Proceedings of the Third International Workshop on Technologies for E-Services*, (LNCS series Vol. 2444) pp. 9-18, 2004.
Nii H. P., "Blackboard Systems. The Blackboard Model for Problem Solving and the Evolution of Blackboard Architectures", *AI Magazine 7(2)*. pp. 38-53, 1986
Corkill D. D., "Blackboard Systems", *AI Expert* 6(9), pp. 40-47, 1991
Erman, L., London, P., Fickas, F., "The Design and an Example Use of HEARSAY-III", Proceedings of the 7th International Joint Conference on Artificail Intelligence, pp. 409- 415, 1981.
Hayes-Roth, B. "A blackboard architecture for control". *Artificial Intelligence*, 26, 251-321, 1985.
Corkill D.D., Gallagher K.Q., Johnson P.M. "Achieving flexibility, efficiency, and generality in blackboard architectures", In *Proceedings of the National Conference on Artificial Intelligence*, pages 18-23, Seattle, Washington, July 1987.
Engelmore R.S., Morgan A., (eds). *Blackboard Systems*. Addison-Wesley, 1988.
Jagannathan V., Dodhiawala R., Baum L.S., (eds). *Blackboard Architectures and Applications*, Academic Press, 1989.
Carver N.. "A Revisionist View of Blackboard Systems". In *Proceedings of the 1997 Midwest Artificial Intelligence and Cognitive Science Society Conference*, May 1997.
Carver, N., Lesser, V. "Evolution of Blackboard Control Architectures", *Expert System with Applications*, Vol. 7, pp. 1-30, 1994.
Canadian Space Agency, "RADARSAT-1 Web page", http://www.space.gc.ca/asc/eng/satellites/radarsat1/default.asp, (last visited 21st November 2007), 2005.
Daniel D. Corkill. "Countdown to Success: Dynamic objects, GBB, and RADARSAT-1", *Communications of the ACM*, 40(5):48-58, May 1997.
Tambouris E. "An Integrated platform for Realising Online One-Stop Government: The eGov Projet", *Proceedings of the DEXA International*

*Workshop "On the Way to Electronic Government"*, IEEE Computer Socity Press, Los Alamitos, CA, p. 359-363, 2001
Bunting D. et al, Web *Services Composite Application Framework (WS-CAF), Ver1.0*, http://www.oasis-open.org/committees/download.php/4343/WS-CAF%20Primer.pdf, (last visited 21st November 2007) 2003
Casati, F., Ilnicki, S., Jin L.J., Krishnamoorthy V., Shan M.C. "Adaptive and Dynamic Service Composition in eFlow". *Proceedings of Advanced Information Systems Engineering: 12th International Conference, CAiSE 2000*, Stockholm, Sweden, pp. 13-31, 2000
Lepouras G., Vassilakis C., Sotiropoulou A., Theotokis D., Katifori A. "An Active Ontology-based Blackboard Architecture for Web Service Interoperability" in *Proceedings of the Second IEEE Conference on Service Systems and Service Management*, China, 2005.
Peristeras V., Tarabanis K., "Reengineering the public administration modus operandi through the use of reference domain models and Semantic Web Service technologies" in *Proceedings of the 2006 AAAI Spring Symposium on The Semantic Web meets eGovernment (SWEG)*, Stanford University, California, USA, Mar. 27-29, 2006.
Loutas N., Peristeras N., Goudos S., Tarabanis K. "Facilitating the Semantic Discovery of eGovernment Services: The SemanticGov Portal" in *Proceedings of the 3rd International Workshop on Vocabularies, Ontologies and Rules for the Enterprise (VORTE 2007) in conjunction with EDOC 2007*, 2007.
KAON development team, *KAON web site*, http://kaon.semanticweb.org/ (last visited November 21st 2007), 2004
Halpin, T. *Information Modeling and Relational Databases: From Conceptual Analysis to Logical Design*. Morgan Kaufmann, ISBN: 1558606726, 2001
Newcomer E. *Understanding Web Services: XML, WSDL, SOAP, and UDDI*, Addison Wesley Professional, ISBN: 0201750813, 2002
Sun Microsystems, *Java™ Remote Method Invocation*, available at http://java.sun.com/j2se/1.3/docs/guide/rmi/index.html (last visited November 21st 2007), 1999
Papamarkos, G., Poulovassilis, A., Wood, P. Event-Condition-Action Rule Languages for the Semantic Web. In *Proceedings of the Workshop on Semantic Web and Databases*, at VLDB 03, Berlin, September 2003.
Wessels D., Claffy K., *Internet Cache Protocol (ICP), version 2 – RFC 2186*, Available through ftp://ftp.ietf.org/rfc/rfc2187.txt (last visited November 21st 2007), 1997.
Wang J., "A Survey of Web Caching Schemes for the Internet", *ACM Computer Communication Review*, (29) pp. 36-46, 1999

Zhu H., Yang T, "Class-based Cache Management for Dynamic Web Content", *IEEE INFOCOM*, 2001
Vassilakis C., Lepouras G. "Controlled Caching of Dynamic WWW Pages", *Proceedings of the DEXA 2002 conference*, pp. 9-18, 2002.
Wessels D. *Squid Internet Object Cache*, http://www.squid-cache.org, (last visited November 21st 2007), 2004.
Date, C., J. *An introduction to database systems (6th edition).* Addison-Wesley publishing Company Inc., 1994. ISBN: 0-201-82458-2.
Freeman, E., Hupfer, S., Arnold, K. *Javaspaces ™ Principles, Patterns and Practice*. Pearson Education, 1999. ISBN: 0201309556
Butenhof, D. Programming with POSIX(R) Threads. Addison-Wesley Professional, 1997. ISBN: 0201633922