UNIVERSITY
OF TRENTO

DEPARTMENT OF INFORMATION AND COMMUNICATION TECHNOLOGY

38050 Povo – Trento (Italy), Via Sommarive 14
http://www.dit.unitn.it

Forum Session at the First International Conference on Service Oriented Computing (ICSOC03)

Marco Aiello, Chris Bussler, Vincenzo D'Andrea, and Jian Yang

Nov 2003

Technical Report # DIT-03-056

Preface

The First International Conference on Service Oriented Computing (ICSOC) was held in Trento, December 15–18, 2003. The focus of the conference — Service Oriented Computing (SOC) — is the new emerging paradigm for distributed computing and e-business processing that has evolved from object-oriented and component computing to enable building agile networks of collaborating business applications distributed within and across organizational boundaries.

Of the 181 papers submitted to the ICSOC conference, 10 were selected for the forum session which took place on December the 16^{th}, 2003. The papers were chosen based on their technical quality, originality, relevance to SOC and for their nature of being best suited for a poster presentation or a demonstration.

This technical report contains the 10 papers presented during the forum session at the ICSOC conference. In particular, the last two papers in the report were submitted as industrial papers.

Trento, December 2003

Marco Aiello
Chris Bussler
Vincenzo D'Andrea
Jian Yang

Contents

Modeling Services and Components in a Service-Oriented Architecture

Zoran Stojanovic, Ajantha Dahanayake, Henk Sol

Systems Engineering Group, Faculty of Technology, Policy and Management,
Delft University of Technology,
Jaffalaan 5, 2628 BX Delft, The Netherlands
{Z.Stojanovic, A.Dahanayake, H.G.Sol}@tbm.tudelft.nl

Abstract. Component-Based Development (CBD) and Web Services (WS) are nowadays used for building flexible enterprise-scale systems organized in a Service-Oriented Architecture (SOA). In order to gain the full benefits of the emerging technology and standards, an effective approach for modeling and architecting this complex distributed computing model is required. Current efforts in this direction are much behind the technology ones. This paper presents an approach to SOA modeling based on the concept of service component and standard UML modeling constructs. The service component interface goes well beyond the simple list of operation signatures in order to specify the complete contract between the service provider and consumer. The paper defines service components of different types, scope and granularity and puts them in the context of a model-driven development process in order to provide bi-directional traceability between business requirements and software artifacts.

1 Introduction

Web services and components have been introduced as promising paradigms for effective enterprise-scale system development and integration. These self-contained and self-describing business-driven functional units can be plugged-in or invoked across the Internet to provide flexible enterprise application integration within the Service-Oriented Architecture (SOA). Basic elements of the new service-oriented paradigm are the standards for interoperability - XML, SOAP, WSDL and UDDI that provide platform-independent communication of software resources across the Internet [16]. On top of that, new languages and specifications for defining the choreography of services that forms real-world business processes are emerging, such as BPEL4WS [4] and WSCI [17]. On the other side of the spectrum, the realization of Web services are usually done using *de facto* standard component implementation platforms - Microsoft .NET and Sun's Enterprise Java Beans, but also traditional programming languages such as C/C++ [10]. Although technology and standards are important in building complex enterprise solutions, they are not sufficient by its own. Besides that, there is a strong need for modeling and design methods and techniques that will guide architecting and building these complex solutions. The reasons for that are manifold. Web services are now becoming more and more an important business

issue and competitive factor among the enterprises. Therefore it is necessary to provide a way to represent services and their choreography at the level of abstraction that is above of XML-based languages such as Web Services Description Language (WSDL) [18], in order to be well-understood by business analysts and users.

Modeling and design of complex service-oriented systems help not only in better understanding of the problem/solution across the project, but also in better communication among involved stakeholders, especially when they are physically separated and/or affiliated to different organizations. Modeling efforts represent a basis of a service-oriented development process that should guide architects and developers in mapping business requirements into software artifacts. The process can provide the quality of a final product, lower the risks in development, deliver solutions on time and help in managing teams of people involved in it. Since technology and standards are constantly evolving, the proper way of modeling and architecting complex Web solutions becomes even more important. Therefore, the new paradigm in system development, called Model-Driven Architecture (MDA) has been proposed [13]. MDA stresses the importance of the platform-independent and platform-specific models to separate abstract domain knowledge from concrete implementation environment. With the development of advanced tools for mapping models into the working code depending on the target platform, the role of models becomes even more important. Using the MDA, models now represent higher-level programming constructs for building business applications that are based on traceable business requirements. Advanced tools can provide automatic generation of XML, Java or other software code based on given models. A proper modeling method should provide a necessary support in deciding what part of the application could be exposed as a service for intra- or inter-organizational purposes, or what part of the system architecture can be realized by invoking a Web service. All these aspects show a great importance of proper modeling and architecting complex Web service-based business systems.

Modeling service-oriented solutions is not a straightforward task. Services inherit many characteristics of their predecessors - objects and components, but also use some elements of workflows and business processes. Therefore, service modeling should cover all these different service facets. Using the standard Unified Modeling Language (UML) as a notation for service modeling is a natural first choice due to its widespread usage [14]. The UML, although originally devoted to object-oriented system modeling, can be easily extended to support modeling of other concepts such as business activities, user interface flows and data schemas. On the other hand, the UML does not still offer mechanisms for representing components and services at the logical level. Service-oriented modeling approaches and development processes hardly exist today. In order to gain the full benefit of new service-oriented paradigm, an approach for conceptual modeling, architecture and design of services, components and related concepts, as well as the mapping of models to software, must be proposed.

The paper presents an approach to SOA modeling using service-based component concepts, interface-based design and the Unified Modeling Language (UML) as starting points. The paper proposes a service component as the main building block of SOA through the set of related concepts, metamodels and granularity types. Business service components are modeled as contract-based service providers to support or-

ganization's business processes through a proper choreography. At the same time they are realized as a composition of lower-level application services and components that can be further mapped to software artifacts. Modeled in this way, the SOA represents a layer of abstraction between business and technology reducing the 'friction' between the two. The paper is organized as following. In section 2, we give the requirements for SOA modeling and present the current support for them in today's modeling methods and notations. Section 3 introduces the concept of service component as the main SOA building block, its main aspects and the elements of its contract. In section 4, we give a modeling approach focused on identifying and specifying the service components of different scope and granularity that in choreography make an e-business solution and provide an effective requirements-to-code mapping. Section 5 concludes the paper pointing out future research directions.

2 Requirements for SOA modeling

Service-Oriented Architecture is an evolutionary, rather than a revolutionary approach. A basis of SOA is the concept of service as a functional representation of a real world business activity meaningful to the end-user and encapsulated in a software solution. SOA provides that loosely coupled services are orchestrated into business processes that support business goals. Similar initiatives were already proposed in the past, such as CORBA services or Microsoft's DCOM. The novelty of SOA is that it relies upon universally accepted standards like XML and SOAP to provide broad interoperability among different vendors' solutions [16]. Even more important is that the level of abstraction is further raised, so that the main building blocks of SOA are now real world business activities encapsulated in the services that offer business value to their consumers.

Due to complexity of service-oriented solutions and diversity of available technology platforms, the MDA approach for designing SOA is a natural choice [13]. Furthermore, modeled components and services in an implementation-independent way represent an abstraction layer between business and technology. Business goals, rules, concepts and processes are captured by components and services at the specification level and further mapped to technology artifacts providing in this way an effective bi-directional traceability between business and technology. The main requirements for SOA modeling on top of current modeling practice are:

- Services and components must be identified and defined in a business-driven way as larger-grained, loosely coupled units that correspond to real business activities and add a measurable business value to their consumers.
- The focus should be on process-driven rather than data-driven service component concepts. Services and components should not correspond to a single business object such as Customer or Order; they should manage information across the set of objects in providing required business functionality.
- Services of the components in a SOA should support particular steps of a business process and should be chained and coordinated in a way to create a business process flow. The modeling focus should be on representing service interaction, nest-

ing, coordination and mutual dependencies, rather than on service internal realization.

- The best practices of interface-based and contract-based design should be applied. The interface of a service component must be extended beyond simple operations' signatures to represent a real business contract between the provider and consumer of the service. Complete and precise, implementation-independent service specification provides effective service discovery and usage.
- It is essential to define different scopes and granularity levels of services fulfilling different roles in business/technical system architecture through recursive composition and choreography. This means that each service can be realized through lower-level services and at the same time is a part of a higher-level service.

The logical starting points for modeling in the SOA are component-based and interface-based modeling and the standard UML. The question is to what extent component-based approaches and the UML can fulfill the requirements above and provide necessary concepts and mechanisms for modeling the SOA, such as componentization, low coupling – high cohesion, interface-based design, hidden implementation, as well as flow of objects and activities.

First-generation component-based development approaches treated components as implementation, binary or source-code software artifacts. That was strongly influenced by the standard UML (version 1.3) where components are defined as implementation diagrams and represented through the implementation diagrams that show aspects of physical implementation [14]. This bottom-up approach to components suggests practicing object-oriented analysis and design and packaging the highly coupled classes/objects into deployable components at the end of the system lifecycle. Such way of thinking certainly results in a semantic gap between objects and Web services, where services are mainly business-driven units rather than low-level programming constructs. Let us see how the standard UML defines some important concepts for SOA modeling, such as *Component*, *Interface*, *Subsystem*, *Collaboration*, *Action* and *Activity*.

After original implementation definition of components, more specification view has been introduced in the UML standard 1.4 and the latest version 1.5, where a component is defined as "a modular, deployable and replaceable part of a system that encapsulates implementation and exposes a set of interfaces." The major revision of the UML (version 2.0), which is expected during this year, promises further improvements in representing components as design-level artifacts. The concept of *Interface* as defined by the UML standard can have only operations. This does not suite well to the SOA needs for representing the outside view of services as comprehensive contracts between providers and consumers. Further improvements are expected in defining contract-based interfaces through specifying operation's pre- and post-conditions using the Object Constraint Language (OCL) [15]. The concept of *Subsystem* in the UML represents a behavioral unit in the physical system; it offers interfaces and has operations, and its contents are partitioned into specification and realization elements. Subsystem concept is a good candidate for representing larger-grained business components. On the other hand, a subsystem has no behavior of its own; it rather packages and exposes the behavior of its contained model elements. This does not fit well in the clear intention that services and components in SOA

represent rich behavioral units. The *Collaboration* in the UML is defined as "a society of classes, interfaces, and other elements that work together to provide some cooperative behavior that is bigger than the sum of all its parts". Collaboration has two aspects: a structural part that specifies the elements of the collaboration and a behavioral part that specifies the dynamics of how those elements interact. In relation to SOA modeling much more emphasis should be put on the concept of Collaboration since it can represent the main architectural building block and serve as a basis for defining a service's internal realization, as well as its usage in the context. Furthermore, the Sequence and Collaboration diagrams that express object interaction must be enriched to support, beside sequencing, other flow and synchronization constructs such as selection (alternation), loops, grouping and parallel tasks. The UML activity diagrams have been already used for business process and workflow modeling [8]. The action semantics is further enriched in the UML version 1.5 and will be further improved in the version 2.0 by more advanced control flow concepts and notations. The new action semantics will provide the UML to be more precise in both the external definition of an interface's responsibilities and the internal definition of its realization within a component, in a form of workflow-based coordination and collaboration of activities provided by finer-grained components and services.

Parallel to the UML evolution, advanced CBD approaches have recognized the benefits of using component concepts as design-level artifacts [1,3,5,7,11]. However, the identification and specification of components are done mainly in an entity-driven fashion, by close matching the underlying business entities such as e.g. Customer, Product, and Order. In this way components are treated more as old-fashioned business objects than business services. The level of service granularity required by SOA corresponds to business component systems in the Business Component Factory method [11] or large system-like components defined in Catalysis [7], but both concepts have not been enough emphasized throughout those methods. The interface of a component in the CBD methods is defined through operations' signatures together with pre-conditions and post-conditions that guard the operations. The methods do not consider extending the interface construct beyond that to offer the ways of expressing e.g. coordination parameters, quality service parameters and configuration parameters that provide much richer component specification.

3 Service Components Concepts

Service-Oriented Architecture in its essence is represented through the choreography of services that provide business added value and are available on the network. Form the point of view of service consumer a service is provided by a component-like entity that hides the service realization details behind the contractual interface. This interface-based view of the service provider actually abstracts its internal details that could vary from collaboration of lower level services and components to monolithic legacy assets and COTS components. It can be said that for each service in the SOA there is a component-like software structure that is responsible for. We assume that being component-like means following the basics of the component way of thinking,

such as principles of “low coupling – high cohesion” and “hidden interior behind exposed interfaces”.

Therefore, for the purpose of modeling the main building blocks of SOA we introduce the concept of *Service Component*. We define a *Service* as a distinct functionality offered by a provider to consumer(s) across the Web with the clear purpose and semantics for both sides. Then, a *Service Component* is an encapsulated, autonomous software entity that realizes and provides the Service through its interface in a contract-based fashion without exposing its internal realization details.

In this paper we focus on the conceptual service and component concepts that represent a foundation of a service-oriented MDA’s Platform Independent Model (PIM) [13]. These concepts can be further extended with particular lower-level technology aspects, such as details of underlying communication protocols and the exact service Web address, and specialized according to the target implementation platform representing a service-oriented Platform Specific Model (PSM). The service component meta-model can be divided into two parts. First part defines the basic concepts describing the very nature of a service component. The second part of the metamodel defines the basic elements of the contract as the main aspect of a service component.

3.1 Basic Service Component Concepts

A service component is a software entity that exists as a distinct, independent, or self-contained unit with a clear purpose of existence. The basic elements of a service component are its context, contract and content (realization). The essence of the service component is the explicit separation between its outside and inside using its contract. A service component does not exist in isolation; it fulfils a particular role in a given context and communicates with it accordingly. The role of the service component is precisely specified by its contract. The contract is realized by the service component realization that is hidden from the context by the contractual interface. The component nature of the service component suggests recursive composition. This means that the service component participates in choreography (i.e. collaborates and coordinates) with other service components to provide a higher-level service to the context, and at the same time each service component can be potentially decomposed into lower-level service components that in choreography realize its contract. The basic service component metamodel is shown in Figure 1.

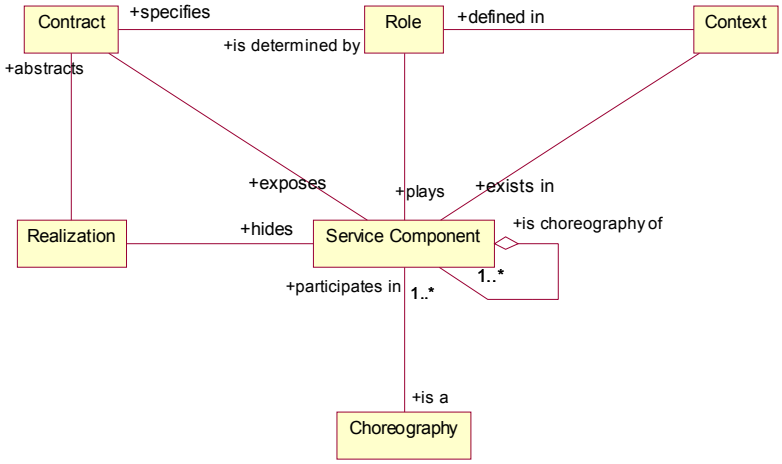


Fig. 1. Service Component metamodel

The component contract is the basic information we have about the component. The contract can be fully specified using different mechanisms, from a human-understandable language, to a formal specification language and to an XML-based language in the case a machine-readable service specification is needed. On the other hand the service component contract can be realized in various ways, as in-house made software code, wrapped legacy system, COTS software assets, or as a composition of lower-level services and components, Figure 2. The way of service realization is not important from the consumer point of view as long as the contract is fulfilled.

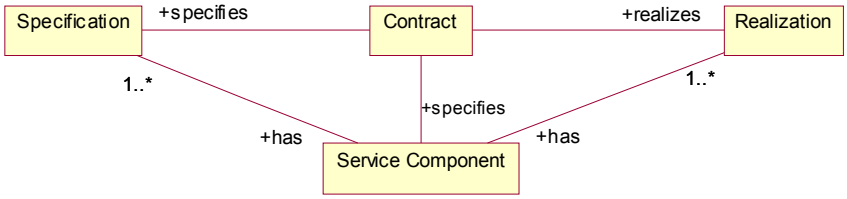


Fig. 2. Specification vs. realization of a service component

3.2 Elements of the Service Component Contract

Fully specified contract elements of a service component represent the complete information about the service necessary for its consumer to use it without knowing its realization details. The contract represents an enriched and enhanced basic interface construct that contains all the information about the service component that must be

known by its context in order to make use of it. In this way a concept of interface goes beyond simple operations' signatures to become a real business contract between a service provider and a consumer.
The following are the basic elements of a service component contract (Figure 3):

Identifier – A service component is identified in the context by its unique name in the naming space with addition of the description of its purpose in the context and the goal of its existence.

Behavior –According to its role(s) in the given context, a service component exposes corresponding behavior by providing and requiring operations and/or by publishing and receiving events to/from its context. Two main types of operations can be defined – operations that perform some computation or transformation (update type) and operations that provide some information on request (query type). They are fully specified in a contract-based manner using pre-conditions, post-conditions, and coordination conditions. Coordination conditions basically specify how provided and required operations, as well as published and received events of a service component are coordinated in time. For precise communication with a service component, not only what operations are provided and required, and how they trigger or are triggered by events, but also how all these activities are correlated in time becomes important for the service consumer to use the service properly. An event from the context that triggers a given operation can be a part of its pre-condition set, while an event emitted by a successful completion of the operation can be a part of its post-conditions. Moreover, provided and required operations of a service component must be correlated in order to determine what operation must be completed before activating the given one, what can be done in parallel or should be synchronized in another way. For example, in the case of the *OrderManager* service component, the operation *MakeOrder* cannot be invoked until the service consumer has not been properly authorized through the *RegistrationManager* service component, or the operation *DeleteOrder* cannot be invoked if the operation *MakeOrder* with the same *OrderID* parameter has not been completed beforehand.

By composing and integrating this coordination behavior of all service components in the given problem context, we can construct a bigger picture of how these service components collaborate and coordinate their activities inside the business process to fulfill the common, higher-level business goal.

Information Types – A service component must handle, use, create or simply be aware of certain information resources in order to provide its services properly. This element of the contract defines what types of information are of an interest to the component, association among them, as well as constraints and rules on the instances of these types. This represents a logical information model of a service component, or its domain vocabulary. In its simplest form, these information types can be considered as type definitions of operations' parameters, as well as types related to them.

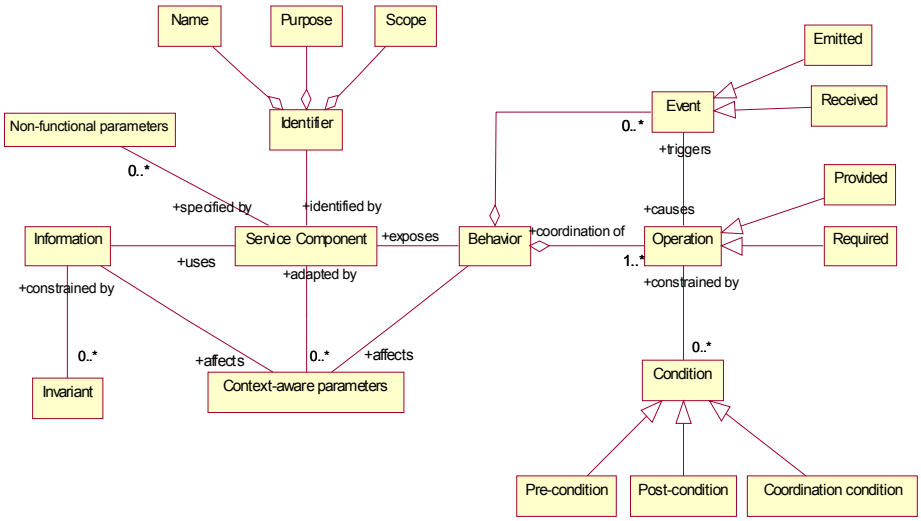


Fig. 3. The metamodel of Service Component Contract

Context-Aware Configuration Parameters – A service component is often dependent on the context of its existence. In order to be used in different contexts or to be adaptable to the changes in its context, the set of configuration parameters should be defined. The example of these parameters could be: required Quality-of-Service (QoS) in a general sense, different profiles of service consumers, and locations in time and space of both service provider and consumer. These parameters can be sent inside the service operation invocation or discovered in another way by the service component in order to adapt its behavior to the changed context situation. Configuration parameters are directly related to the realization of service operations by customizing and adapting it accordingly. The concept of configuration context-aware parameters is an important step towards intelligent, self-adaptable services.

Non-functional (Quality) Parameters - A service component can define a set of so-called non-functional parameters that characterizes the "quality" of its behavior in the context. These parameters can be important decision elements for the service consumer to decide whether to use this service or find another one with the same or similar contract. Non-functional parameters are for example Performance, Reliability, Fault Tolerance, Cost, Priority, and Security.

The Table 1 presents how the contract elements can be expressed using available UML constructs as well as how they can be mapped to the WSDL elements.

Table 1. Mapping between Service Component Contract Elements, UML constructs and WSDL elements

Contract elements	UML artifacts	WSDL elements
Service Component	Stereotype of a Subsystem or a Class	Service
Name	Name - textual	The Name tag in the definitions element
Purpose, Goal	UML Notes	Documentation element
Operations - provided and required	Interface with signatures of operations	Operations, messages, Port types elements
Events – published and recieved	Not supported	Not supported
Pre-conditions, post-conditions	Object Constraint Language (OCL) grammar attached to operations	Not supported
Coordination conditions (Choreography of operations and events)	Activity diagrams; or sequence diagram enriched with action semantics	Not supported (WSCI can support this)
Information types	Stereotyped classes	Types element
Information dynamics, the changes of data instances through time	State-charts	Not supported
Invariants and constraints on information types	OCL grammar	Not supported
Context-aware configuration parameters	Properties of a Subsystem or a Class that represent a Service Component	Not supported
Non-functional parameters	Properties of a Subsystem or a Class that represent a Service Component	Not supported
Technical details (protocols, network addresses, data encoding styles)	Specified in the MDA's PSM	Binding element

Contract elements defined above specify conceptual, technology-independent aspects of a service component according to the MDA's PIM. In the sequel of the development process, contracts will be extended with the specification about concrete service bindings, such as data encoding styles, transport protocols and service network addresses to provide a basis for service realization. It can be concluded that the WSDL as the language for service description must be extended to support an important elements of the service component contract defined above. At the same time various elements of the UML stereotyped and combined properly can serve the pur-

pose of representing given service component concepts in the diagrammatic form at the level above of the XML-based grammar.

4 Service-Oriented Modeling

One of the important modeling and design issues in building SOA solutions is to determine the type, scope, and granularity of service components necessary for representing the main architectural abstractions. Moreover, these different types of service components must be put in the wider context of a development process to provide that traceable business requirements are mapped to software artifacts supporting them.

4.1 Component Types

Although different types of service components having different scope and granularity can exist in the complex SOA, two basic service component types - Business Service Component (BSC) and Application Service Component (ASC) can be defined as shown in Fig. 4.

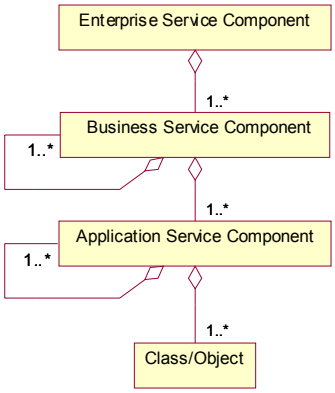


Fig. 4. Component types

A BSC provides services and operations that have meaningful, perceivable and measurable value in the business context; it realizes specific part(s) of the business process and in collaboration and coordination with other BSCs fulfills an overall business goal. On the other hand, an ASC provides finer-grained operations that do not offer a real business value; it collaborates and coordinates with other ASCs to provide a business behavior represented by a BSC that encapsulates them. A BSC can represent a collaboration of other BSCs and ASCs. An ASC can represent a collaboration of other ASCs and lower-level elements such as objects/classes.

On the opposite side of the spectrum, a coarse-grained BSC that exists autonomously in the enterprise, provides a complete marketable value and communicates

with other elements of the enterprise (people or automated systems) is called enterprise service component. Functionality offered by a BSC can be used as both inter- and intra-enterprise Web service in a SOA. On the other hand, an ASC can be mainly used as a Web service in intra-enterprise settings.

By focusing on these basic service component stereotypes we can define two levels of a service-oriented Platform Independent Model. The first level of the PIM defines how a business process involving different partners across the Web is supported through contractual collaboration and coordination of business service components that realize the particular steps of the process. The second PIM level "opens" black-box BSCs and defines how their internal design is realized through collaboration and coordination of finer-grained ASCs mainly inside an enterprise.

4.2 Requirements-Architecture-Code Traceability

This section puts service component types in the context of an architectural modeling approach, which should provide that the designed SOA is based on traceable business requirements and can be straightforwardly mapped to software artifacts. The best way to design SOA is the combination of a top-down and bottom-up approach with small increments both back and forward. However, initial identification of BSCs must be made in a top-down fashion, since BSCs have to capture real business issues, goals, activities and rules. The requirements-to-code traceability is further provided by decomposing BSCs into finer-grained components and services, while at the same time taking into account already existing software assets.

For the purpose of BSC identification, first domain object analysis as well as use case analysis should be performed. Domain object analysis defines a domain vocabulary of the system being developed, i.e. information about business concepts in the problem area that should be handled by the system together with their attributes and relationships. Use case analysis is an effective mechanism for defining cohesive sets of features and services on the system boundary since they capture the intended behavior of the system without having to specify how that behavior is implemented [12]. The use cases of the proper granularity correspond to user business goals and activities, i.e. Elementary Business Processes (EBPs) as suggested in [6]. So-called change cases as potential changing requirements on the system in the future can be included as well to provide higher adaptability of the system solution in the future.

Use cases can be specified in details according to the use case template that includes name, description, involved actors, goal in the context, scope, level, pre-conditions, post-conditions, triggers, main success scenario, extensions, sub-scenarios, priority, frequency, performance, etc. [2,6]. This use case specification can be extended with the information about the role of the use case in a wider context of a business processes given system should participate in. Therefore for each use case, the use cases that precede it, follow it, perform in parallel with it or are synchronized in other way with it should be defined. Furthermore, for each use case, its superordinate and subordinate use cases should be defined providing a composite hierarchy of use cases, i.e. corresponding business goals. This can be illustrated using an activity diagram with use cases represented as action states of the diagram, or using a se-

quence diagram enriched to express the action semantics (sequence, selection, loop, fork/join, etc.) with the use cases on the horizontal axis of the diagram. Finally, domain information types resulted from domain analysis are cross-referenced with the use cases. In this way for each use case what information types are needed for its performance are defined. Finally, the first-cut BSCs are defined as providing operations that support the realization of one or several cohesive business goal–level use cases, Figure 5.

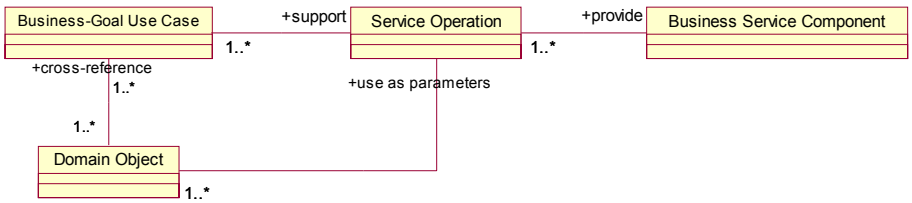


Fig. 5. Relation between the concepts of use cases, domain objects, service operations and business service components

In order to decide what use cases are under the responsibility of a particular BSC, and how to group use cases into distinct BSCs, a number of business and technical criteria must be taken into account:

- Use cases that handle the same domain information objects by creating, updating or deleting them should be under the responsibility of a single BSC.
- Use cases that are expected to change in the same rhythm and under the same circumstances should be under the responsibility of a single BSC.
- Use cases already supported by an existing solution (legacy, COTS or web service) used in the project should 'belong' to a single BSC that models the given solution.
- Use cases that belong to physically distributed parts of the system cannot 'belong' to the same BSC that basically represents a unit of collocated services.
- Use cases that belong to the same business transaction or correlated transactions required or performed together should be under the responsibility of the single BSC.
- Use cases that coherently represent a known marketable and attractive unit of business functionality and therefore can be successfully exposed as a Web service product, should 'belong' to a single BSC.

The main criticism of the use case analysis is that it can lead to the functional decomposition of the system [2]. By limiting our scope to business goal-level use cases we ensure that our BSCs offer real business functionality, not low-level technical functionality. Moreover, by coupling use cases based on information objects they handle (the first item of the list above) we provide a balance between process-based and entity-based decomposition of the system. The specification of a BSC is now done (along the elements defined in the section 3) based on the specification of use case(s) the BSC supports, and domain information types these use case(s) handle. All the necessary elements for this specification can be derived from the use case tem-

plate specification with coordination extensions as mentioned above together with the specification of information types, their attributes, relationships and constraints. In creating an activity diagram based on identified use cases, it must be ensured that the pre-conditions of a use case should match the post-conditions of the use that precedes it. In this way the whole business process can be constructed in the form of an activity diagram representing the flow of activities and objects between use cases.

Now when we have defined BSCs, a service developer would like to design their interior. The contract of a business component is realized in terms of collaboration and coordination of finer-grained ASCs. Since BSCs are defined across the logical tiers of the network, the following types of ASCs can be defined according their roles in supporting the BSC contract realization:

- Consumer Interface ASC – provides services necessary for communication with the consumer of the BSC in terms of transforming consumer's input into the form understandable by the service logic, and the service output into the form understandable by the consumer. This ASC makes the service logic available to different types of consumers. If a consumer is a human user, corresponding presentation logic is added to the ASC.
- Data Access ASC – provides services for accessing data stored in a permanent storage. The role of this ASC is to protect the BSC logic from peculiarities and possible changes of the underlying database structure. Depending on characteristics of the data storage, this ASC can be enriched with proper access methods (indexing structures) that speed up information retrieval, as well as with mechanisms for data mining.
- Business Logic ASC – provides the services that determine the logic of the BSC. These ASCs can be further divided into two basic types based on the type of services they provide (query or update):
  - ASCs that mainly provide information about particular business entities, i.e. represent the so-called contact point for that information and
  - ASCs that encapsulate computation logic, control logic or data transformation logic that are not parts of any business entity object and can possibly handle several business entity objects. Since a BSC is defined as a realization support for one or several cohesive use cases, the ASCs that realize included and/or extended use cases of those business-level use cases, can be identified as well.

One of the essential elements of each BSC is a *Coordination Manager* service component. The main role of this component is to manage and control how the ASCs interact to produce a higher-level business goal specified by the contract of the BSC that encapsulates them. A Coordination Manager can be realized using the combination of the Mediator and Façade design patterns as defined in [9]. In this way, choreography aspects and flows of controls and activities between ASCs inside the given BSC is encapsulated and managed by a single software unit. This mechanism hides the complexity of internal choreography of the BSC and at the same time provides better flexibility of the solutions and replaceability of particular ASCs.

Fully specified business and application service components, as well as their interactions and choreography represent a service-oriented platform-independent model. The model of the service components that should be built in-house can be first trans-

formed into platform-specific model using the corresponding UML profile and finally implemented in the target technology platform. This model-to-code mapping can be provided by today's advanced tools in the form of code generation and round-trip engineering. Provisional mapping of different aspects of service components presented here and Enterprise Java Beans constructs is shown in Table 2. Further details are out of scope of this paper.

Table 2. Mapping logical architecture concepts to implementation constructs

Service Components	**EJB constructs**
Consumer Interface ASCs	Servlets, Message-Driven Beans, Java plug-ins
Data Access ASCs	Database APIs, JDBC
Computation logic ASCs	Session Beans, Message-Driven Beans
Information provider ASCc	Entity Beans
Data types	Java classes
Coordination Manager	Session Beans

5 Conclusion

Although there has been a huge interest in Web services paradigm for truly interoperable inter- and intra-enterprise application development and integration during the last several years, little attention has been paid to the approaches for modeling and architecting this complex computing model. Current achievements in this respect are much behind the technology ones where new standards and languages for interoperability are constantly emerging.

In order to gain the full benefit from practicing Service-Oriented Architecture (SOA), further efforts must be made in defining corresponding modeling and design approaches focused on the concept of service. SOA modeling is a challengeable task since the service concept further raises the level of abstraction and actually aims at finally closing the gap between business and software. Therefore, for the purpose of SOA modeling, the best practices from object-orientation and component-based design, but also workflow and business process modeling must be taken into account and properly integrated. Although there is established CBD methodology practice that covers certain aspects of the modeling and architectural design of component-based solutions, the SOA modeling poses certain requirements on top of it. Therefore, straightforward applying of existing UML and CBD concepts for the purpose of modeling the SOA, although a good starting point, is not a feasible approach. The UML component concept as a natural basis for SOA modeling is still mainly implementation-related, while popular CBD methods are mainly focused on finer-grained entity-driven components.

The aim of the paper is to propose a way to model services, components that realize them and their choreography, as the main building blocks in a SOA. The paper introduces the concept of service components that represent a paradigm shift from components as objects to components as service managers that fits well to SOA modeling. The interface specification of the service components is extended to provide the

comprehensive contract between service provider and consumer. The service component contract is defined using the UML constructs, which is at the level of abstraction above XML-based languages such as WSDL. The contract expressed in the UML provides better understanding and communication among the project stakeholders, and at the same time can be easily mapped to the WSDL constructs and furthermore to corresponding software code. The two basic types of service components are defined, namely Business Service Components and Application Service Components. These components are the basic elements of two-level MDA's PIM that provides the effective bi-directional mapping between business requirements, logical service-oriented system architecture and software assets. Further steps include the definition of a precise UML Profile for service-oriented modeling, as well as the mapping rules for transforming platform-independent models into platform-specific models according to the target development platforms and technology standards.

References

1. Apperly, H. et al.: Service- and Component-Based Development: Using the Select Perspective and UML. Addison-Wesley (2003)
2. Armour, F. and Miller, G.: Advanced Use Case Modeling. Addison-Wesley (2001)
3. Atkinson, C., et al.: Component-Based Product Line Engineering with UML. Addison-Wesley Publishing (2001)
4. BPEL4WS – Business Process Execution Language for Web Services, information available at ftp://www6.software.ibm.com/software/developer/library/ws-bpel11.pdf
5. Cheesman, J. and Daniels, J.: UML Components: A Simple Process for Specifying Component-Based Software. Addison-Wesley (2000)
6. Cockburn, A.: Writing Effective Use Cases. Addison-Wesley (2001)
7. D'Souza, D.F. and Wills, A.C.: Objects, Components, and Frameworks with UML: the Catalysis Approach. Addison-Wesley (1999)
8. Dumas, M., ter Hofstede, A.H.M.: UML Activity Diagrams as a Workflow Specification Language, UML 2001, LNCS 2185, pp. 76–90, 2001.
9. Gamma, E., Helm, R., Johnson, R. and Vlissides, J.: Design Patterns: Elements of Reusable Object-Oriented Software. Addison Wesley (1995.)
10. gSOAP: C/C++ Web Services and Clients, Information available at http://www.cs.fsu.edu/~engelen/soap.html
11. Herzum, P. and Sims, O.: Business Component Factory: a comprehensive overview of business component development for the enterprise. John Wiley & Sons (2000)
12. Jacobson, I., Christerson, M., Jonsson, P., and Overgaard, G.: Object-Oriented Software Engineering: A Use Case Driven Approach. Reading, MA: Addison-Wesley (1992)
13. MDA- Model Driven Architecture, OMG (Object Management Group), Information available at http://www.omg.org/mda/
14. UML- Unified Modeling Langauge, OMG (Object Management Group), Information available at http://www.omg.org/uml/
15. Warmer, J.B. and Kleppe, A.G., "The Object Constraint Language (OCL): Precise Modeling with UML", Reading, Mass., USA: Addison-Wesley, 1999.
16. W3C. World-Wide-Web Consortium: XML, SOAP, WSDL. Information available at: http://www.w3c.org/
17. WSCI – Web Service Choreography Interface – Available at http://www.w3.org/TR/wsci/
18. WSDL –Web Services Description Language, Available at: http://www.w3.org/TR/wsdl12/

Polyarchical Middleware for On-Demand and Multi-Standard Services' Composition for Ubiquitous Computing

M. Yu, A. Taleb-Bendiab, D. Reilly, E.Grishikashvili, Wail Omar

School of Computing and Mathematical Science
Liverpool John Moores University
{cmsmyu; a.talebbendiab; d.reilly; cmsegris; cmpwomar}livjm.ac.uk

Abstract. Whilst the vision of on-demand computing is very seductive it engenders its own technical challenges including; design, development and deployment of ubiquitous utility services, and low-cost, low-skills and low-latency services re-assembly for lifetime management runtime due to complex and unpredicted service discovery, interoperation and adaptation. In this paper we argue for the need of a new model for on-demand ubiquitous services' activation through a polyarchical middleware, which enables on-demand composition of software applications regardless of service standards and middleware used. This paper will present early results of a research study into the development of a framework for ubiquitous service invocation/activation, which provides an abstract model for on-demand ubiquitous service composition and execution. This proposed polyarchical middleware could be used for on-demand wireless application service composition, which include ad-hoc service discovery, assembly using virtual containers, invocation and adaptation. The paper will finish with a critical review of our model and concluding remarks followed by an indication of further work

1. Introduction

Static distributed service composition has gradually presented limitations under a dynamically changeable network environment. The critical runtime usage requires distributed systems to be capable of automatically adapting and/or evolving their behaviours in response to any unpredicted changes such as: network faults, requirements changes and/or failed services. We feel that runtime services' composition can be more flexible than the traditional static approach. In order to best-fit end-users' needs, services can be composed/constructed at runtime from a set of available network and software services.

However, such dynamic composition must address many technical issues including: seamless integration/interoperation, failure-detection and robust self-healing abilities. In order to respond to such runtime requests, distributed middleware solution has been designed to bridge the gap between the distributed components and on-demand service application. Through a higher-level programming abstract model,

middleware architectures such as DCOM [1], CORBA [2] , Jini [3], Web services [4], UPnP [5] provide different programming models for service composition and adaptation at both design and runtime.

However, such middleware technologies require that components must implement a single programming standard (e.g. Java Remote Method Invocation technology [6]) or adhere to a certain definition (i.e. CORBA Interface Definition Language [7]). If service composition occurs within the same standard, these distributed middlewares can still flexibly respond to runtime on-demand requests. But considering the critical reality, service components may have been developed and deployed using different standards and even different operating systems. The current distribution middleware models are in general unsuitable for service composition and adaptation between different component standards. In short, such multi-standard service composition and activation would require a tremendous amount of manual configuration and adaptation. Ideally, if services of one system could be automatically discovered by another system and utilized, it would dramatically enhance system on-demand service composition and adaptation abilities at runtime. Without redesigning the system to accommodate the unpredicted changes, or even with less intervention from users, the distributed system can reconfigure/update the services without disrupting the system at runtime.

The next generation of distributed architectures--Ubiquitous/Grid computing [8, 9] have been promoted to offer end-users a large scale virtual robust computing infrastructure. Within such a higher service composition model, components from different organizations and locations can seamlessly work together to solve a specific problem/request as mentioned in Open Grid Services Architecture (OGSA) [9]. In this paper, we describe a *polyarchical* middleware, which extends core middleware service models (of both Object-Oriented and Message-Oriented middleware) to provide on-demand multi-standard services' composition and adaptation across ubiquitous computing playforms.

Our model is based on the service-oriented architecture, in which service location transparency is addressed by services that can be dynamically discovered and invoked. Such invocations may come from any other user application, components and/or software services whilst, the technical details of service composition and adaptation will be hidden away from end-users.

We address the challenge of runtime invocation adaptation to support service composition and self-healing strategies. Accompanied with a well-defined *Assembly Service Description Languag*e, dynamic configuration and self-adaptation can be exactly executed after precisely knowing about the environmental circumstances of service components. In our Assembly Services Description Language, we offer more information about the Services/Service providers, which are needed by a Service Composition Manager to locate the "best-fit" service components at runtime.

We concentrate on enabling automatic service discovery and invocation, in which case data and service information are dynamically incorporated into the invocation code. Human intervention is only essential to set up on-demand service requests and possibly to handle major changes in the environment or unexpected exceptions.

The reminder of the paper is organized into six sections: Section 2 describes several ongoing related research and development themes; Section 3, describes our dynamic service composition solution, used to assemble on-demand service applications.

Section 4, outlines our proposed service adaptation strategy, used to compose service components and enable self-healing across varieties of standards/architectures; Section 5, considers the development of the Polyarchical Middleware architecture, which mediates the on-demand service requests and hides the technical details. Section 6 describes a recent case study, used for on-demand wireless application service composition. Finally, Section 7 draws overall conclusion and mentions directions for future work.

2. Related Work

The component-based software engineering community has widely used the connector and proxy model to model and abstract inter component interaction and invocation [10].

Web Service Invocation Framework (WSIF) [11], is designed to supply users with simple APIs to invoke Web services through abstract representations of the services, rather than the usual Web Services programming model of working directly with the SOAP APIs. So far, WSIF provides a degree of protocol-independence, but however it currently assumes Simple Object Access Protocol (SOAP) as the underlying protocol. It allows developers/users to adopt own programming models without worrying about the service implementation regardless of SOAP packages used behind. However, our protocol-independence is designed to cross multiple service standards rather than just SOAP.

Open Grid Services Architecture (OGSA), still in the early stage, has been introduced by IBM to integrate *Grid*-enabled *Web services* across the Internet. The transport protocol SOAP is used to connect data and application on Grid service discovery, sharing and composition in a dynamic distributed environment. The standard interface of Grid service, described by Web Service Description Language (WSDL) [12], includes multiple bindings and implementations. Such Grid services can be deployed on different hosting environments-even different operating systems. As one of the invocation proxies, the existing WSIF has been used to dynamically detect the SOAP binding protocols and construct the appropriate invocation code. Whilst, rather than only trying to deal with the Grid-enabled Web services, we have deliberately concentrated on ubiquitous services deployed under different service standards and core service middleware models (Object-Oriented and Message-Oriented middleware).

OpenWings [13] was first set up to meet the military use to discover replacement services in a critical and volatile environment. OpenWings uses a component-connector model to represent the inter-communication between two different standards. By creating a protocol abstract layer, both sides need to implement special APIs to translate method calls to and back from that layer, where all the data and information will be transmitted. Our work has a similar overall goal but has focused on runtime service adaptation code, which requires the generation of extra files. Both sides can still retain their original design without having to implement any special APIs or sender and receiver proxies.

Our own approach has several similarities to the research work described above, but it concentrates specially on the use of the middleware service model to provide the assembly of applications in an ad-hoc manner [14, 15]. This ad-hoc assembly is based on the users' requirements across ubiquitous computing platforms. In particular, our Assembly Services Description Language approach offers additional information about the services/service providers to that normally provided in UDDI and WSDL strategies. This additional information is used by the service composition manager to lookup and determine the "best-fit" matching services at runtime. Our dynamic configuration and self-adaptation strategy across multi-standard services actively contributes towards maintaining the service workflow process at runtime. It does so by dealing with any dynamic changes at runtime, such as service failure, upgrades, replacement and/or evolution. Through this strategy our middleware model also enables the full utilization of the existing multi-standard service resources, without forcing both client and server sides to implement special APIs. Even when new service component standards are introduced in the future, our middleware is capable of quickly adapting to utilize the new standards with minimum reconfiguration.

3. Dynamic Service Composition

Unlike the traditional predictable and repetitive processes, end-users can compose/assemble their own on-demand services at runtime from a range of services providers over the network. However, these services are likely to be developed and deployed using varieties of standards and middleware technologies.

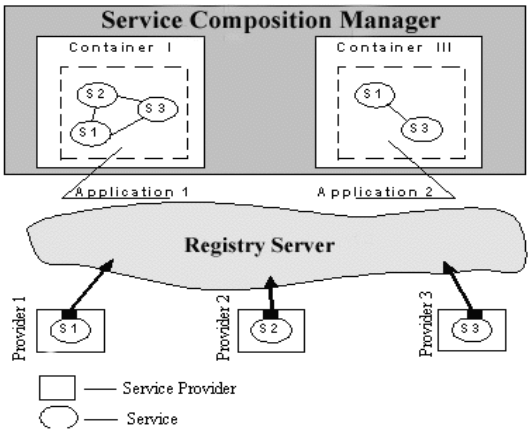
In this paper, we introduce a dynamic service composition approach, which uses service virtual containers and well-defined Assembly Services Description Language to achieve runtime dynamic service composition and/or self-healing.

3.1 Service Virtual Container

A *Service Virtual Container* is a virtual environment in which services are assembled ready for composition into an on-demand application/service. Normally, each *Container* may contain one or more services as shown in Figure 1. A *Service* is a logical concept, which could be devices or software applications distributed over the network. These *Services* reside in varieties of existing physical containers (e.g. Java Virtual machines, Web server containers). The Service Composition Manager middleware service is responsible for the actual dynamic service composition. Any runtime service discovery, invocations and failure exceptions will be detected and handed over to other managers for them to cope with in the proposed framework.

A *Service Provider* is a set of all necessary class files to enable service publication and service invocation over the network via a certain protocol such as Java RMI, SOAP/RPC and/or SOAP/messages. Service providers enable services to be discovered and accessed dynamically by software applications or other service providers through a specific service protocol.

Fig. 1. Service composition manager



In Figure 1, *Provider 1* offers *Service 1* by registering it on a specific registry server (such as a Java RMI registry server, a Jini lookup server or a Web service UDDI server) or our local Registry Server using its Assembly Services Description Language (ASDL). The latter[1] is an XML-based document containing information about the service, which is needed by software applications or other service providers to access or invoke the service. On the first invocation of a service method, the service comes to life within the *Virtual Service Container*. Later, more matched services may be added into the virtual container to meet the demands of end-users.

3.2 Assembly Services Description Language

We feel that most types of existing service registry servers, such as Java RMI's registry server, Jini's Lookup server and CORBA's Naming server, are insufficient to support service description functionality for their registered service components. Even the current Universal Description, Discovery and Integration (UDDI) register server, associated with WSDL, is still not enough to describe the services using types and version of web services description language, methods, input and output parameters, dependencies, etc. Through our own Assembly Services Description language, we have tried to provide more information about the services, which are needed by software applications or other service providers to decided which one of the available services is more suitable and how to access or invoke it. The Assembly Services Description Language (ASDL) describes all the Meta-information that is related to services. This Meta information will be used for the deployment and discovery of the services as well as the invocation of their methods. Figure 2 shows a diagram of ASDL, follow by illustrative example using XML format shown in figure 3.

[1] The difference between ASDL and WSDL associated with UDDI will be discussed in the next section.

Fig. 2. A Diagram of ASDL

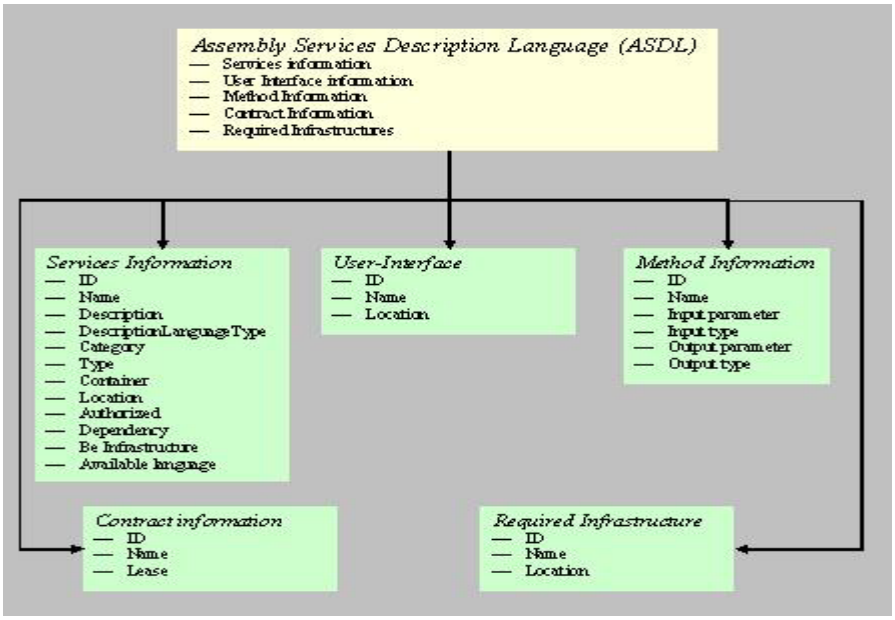


Fig. 3. Assembly Service Description Language excerpt: (a) Main Service Description, (b) WSDL Service Description

```
<Service ServiceID="S1">
     <ServiceName> Calculator </ServiceName>
     <ServiceDescription> http://localhost:8080/userdoc/Calculator.wsdl </ServiceDescription>
     <ServiceCategory>Timer</ServiceCategory>
     <ServiceType>Research</ServiceType>
     <Container>cmsgris</Container>
     <ServiceLocation>cmsgris:8080/userdoc/Calculator</ServiceLocation>
     <ServiceBindingType>Java RMI</ServiceBindingType>
     <NumberOfUserInterface>1</NumberOfUserInterface>
     <InterfaceName>TimerSet</InterfaceName>
     <InterfaceLocation>http://cmsegris:8080/userdoc/</InterfaceLocation>
     <NumberOfMethod>1</NumberOfMethod>
     <Method>SetTimer</Method>
     <Dependency>Antecedent </Dependency>
     <Authorized>True</Authorized>
     <ServiceContract id="SC1">
          <ServicesContractName>SCCmsgris</ServicesContractName>
          <ServicesContractLease>1/1/2004</ServicesContractLease>
      </ServiceContract>
      <AvailableLanguage>Eng,Fr,Ar</AvailableLanguage>
      <ServicesType>Research</ServicesType>
      <RequiredInfrastructure>
           <InfrID>I1</InfrID>
           <InfrID>I2</InfrID>
       </RequiredInfrastructure>
       <BeInfrastructure>True</BeInfrastructure>
</Service>
```

```
<?xml version="1.0" encoding="utf-8"?>
<Service >
    <ServiceName>Calculator</ServiceName>
    <ServiceDescription>Do the basic Arithmetic Operation
     </ServiceDescription>
    <DescriptionLanguageType>WSDL1.1</DescriptionLanguageType>
    <ServiceCategory>Basic Calculator</ServiceCategory>
    <ServiceType>Research</ServiceType>
    <Container>cmsgris</Container>
    <ServiceLocation>cmsgris:8080/userdoc/Calculator</ServiceLocation>
    <UserInterface id="UI22">
            <UserInterfaceName>CalcInterface</UserInterfaceName>
            <UserInterfaceLocation>http://cmsegris:8080/userdoc/
            </UserInterfaceLocation>
    </UserInterface>
    <Method id="M22">
            <MethodName>Addition</MethodName>
            <InputParameter>a,b</InputParameter>
            <InputType>Integer,Integer</InputType>
            <OutputParameter>c</OutputParameter>
            <OutputType>Integer</OutputType>
    </Method>
    <Method id="M23">
            <MethodName>Multiplication</MethodName>
            <InputParameter>a,b</InputParameter>
            <InputType>Double, Double</InputType>
            <OutputParameter>c</OutputParameter>
            <OutputType>Double</OutputType>
    </Method>
    <Dependency>Antecedent </Dependency>
    <Authorized>True</Authorized>
    <ServiceContract id="SC1">
            <ServicesContractName>SCCmsgris</ServicesContractName>
            <ServicesContractLease>1/1/2004</ServicesContractLease>
    </ServiceContract>
    <AvailableLanguage>Eng,Fr,Ar</AvailableLanguage>
    <ServicesType>Research</ServicesType>
    <RequiredInfrastructure>
            <InfrID>I1</InfrID>
            <InfrID>I2</InfrID>
    </RequiredInfrastructure>
    <BeInfrastructure>True</BeInfrastructure>
</Service>
```

Assembly Service description language: we consider runtime dynamic service composition as a challenging issue, especially within such crucial time limits. As we have found, it is hard to precisely predict the exact environmental circumstances of service execution, and whether the process will be successful at composition time rather than design time. We also noticed that in practice not every discovered service could be used as a service composition component. Often, the required on-demand service composition is significantly different from available service components. With these point in mind, we feel that it is important to have a well-defined service description for each component, which describes functionality as well as execution environment. The service description can greatly enhance the runtime service discovery process and assist in finding the best-fit components for service composition. Meanwhile, it can also reduce the system response time, since less intervention is required from end-users. Figure 3 shows an example of an XML-based *Service Description Language file*, which is used herein to support runtime service discovery and composition.

Unlike the WSDL, associated with traditional UDDI, an ASDL includes not only component information, such as service method name or message parameters, but also the infrastructure, which provide the operational containers for software components.

The ASDL uses a tag *ServiceDescription* (Fig. 3) to refer to an external service description document, which can be encoded in WSDL or any other available description language technology. The *RequiredInfrastructure* tag describes the infrastructure (Fig. 4), which provides the execution environment for the service component.

Fig. 4. Infrastructure description excerpt

```
<Infrastructure infrastructureID="I1">
    <infr_Name>Addition</infr_Name>
    <infr_Description>To do the addition process</infr_Description>
    <infr_Type>Software</infr_Type>
    <infr_Container>cmswomar</infr_Container>
    <infr_Host>www.jmu.ac.uk</infr_Host>
    <infr_Server>Apache/1.3.0 (Unix)</infr_Server>
    <infr_Platform>any</infr_Platform>
    <infr_Middleware>.Net</infr_Middleware>
    <infr_RequiredProcessor>PI 233MHz</infr_RequiredProcessor>
   <infr_RequiredMemory>8Mbyte</infr_RequiredMemory>
   <infr_RequiredStorageSize>1.4MB</infr_RequiredStorageSize>
   <infr_Contract id="IC1">
   <infr_Contract_Name>cmpwomar</infr_Contract_Name>
        <infr_Lease>3/7/2004</infr_Lease>
  </infr_Contract>
  <infr_Catogery>Addition</infr_Catogery>
</Infrastructure>
```

The details of service components for both software and infrastructure provided in ASDL can greatly assist in the discovery/lookup of the right service components for composition at runtime.

4. Service Adaptation

A successful service composition requires seamless service interoperation and well self-healing to recover from unexpected failures/changes. Both these strategies require robust service adaptation to provide backend support between different service standards. In this section, we describe a service adaptation mechanism based on runtime code generation, which renders our approach different to those considered previously in the related works.

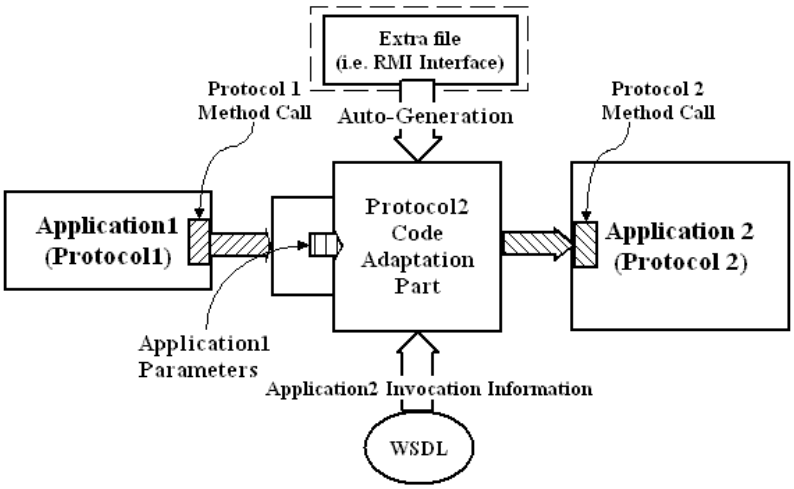
4.1 Service Adaptation Mechanism

We begin by considering a scenario, in which two applications, based on different programming models (e.g. client-server/peer-to-peer) need to communicate over a network. For such communication, our approach makes use of runtime auto-generated service adapters. The adapters can dynamically provide the necessary code adaptation

to handle service interoperation and integration among different standards. The adapters make use of a template file, in which the invocation code for each individual standard has already been specified. These adapters alleviate the need to implement any special APIs or proxies or classes on both sides. The adapters also set our work aside from that conducted by Openwings and WSIF. The use of service adapter means that developers and end-users can ignore the details of the service adaptation and simply stick to their original implementation mechanisms. The following diagram in Figure 4 shows the anatomy of service interoperation sequence.

The protocol 1 method call can be reset to the normal protocol 2 call mechanism after code adaptation. Application 2 can still be invoked by the protocol 2 method call using the normal data encoding (i.e. serialization/deserialization in Java RMI or marshalling/unmarshalling in SOAP-RPC of parameters and results). In contrast to our own work, other researchers, including OpenWings, intend to translate method calls into a standardized protocol transmission.

Fig. 5. Service interoperation sequences



Invocation information is retrieved by parsing a service WSDL document and any extra files such as stub and/or proxy classes (i.e. Java RMI Interface) is also auto-generated at runtime.

4.2 Adaptation Scenario

Here we consider two scenarios to demonstrate service adaptation behaviours. The first scenario represents default behaviour in that it attempts alternative invocations, regardless of service standard/protocol, without the need for end-user/system intervention. The second scenario considers how the user/system may specify *particular preferences* relating to service standards or middleware types to be used in invocations.

Default behaviour: Assumes end-users/systems have no particular preferences relating to the type of middleware technology or service standard used by the alternative service components. When a service is found (via lookup) in the registry server[2], the Polyarchical Middleware initializes the invocation adapter template file to access/invoke any matched services as possible alternatives with the adaptation details hidden from end-users / systems. If a service invocation should fail, an exception will be caught and the next alternative component and associated connector will be sought and tried automatically, without end-user/system intervention.
Particular Preferences: allows the user/system to select specific service standard(s)/middleware technology(s), as options, which may serve as special runtime requirements. End-users/systems may still take advantage of the polyarchical middleware when it comes to the access/invocation of their own services. To do so, they first register their specific service and then allow the polyarchical middleware to take care of any adaptation of the invocation mechanism used to access/invoke their service.

5. Framework Structure

The Sun™ Open Network Environment (Sun ONE [16]) first used the term *Polyarchical* to describe features of the next generation distributed architecture. Polyarchical middleware intends to provide smarter and better communication facilities to allow end-users to set up distributed applications using a higher–level model. Our own polyarchical middleware was implemented in Java language. Java was selected due to its dynamic features and support for web-based applications. Java also has the advantage of platform independence and provides utilities for parsing and interpretation of description languages. However, we would like to think that ideas underlying our polyarchical middleware are not wholly dependent on Java and they may be implemented using other languages or even language combinations.

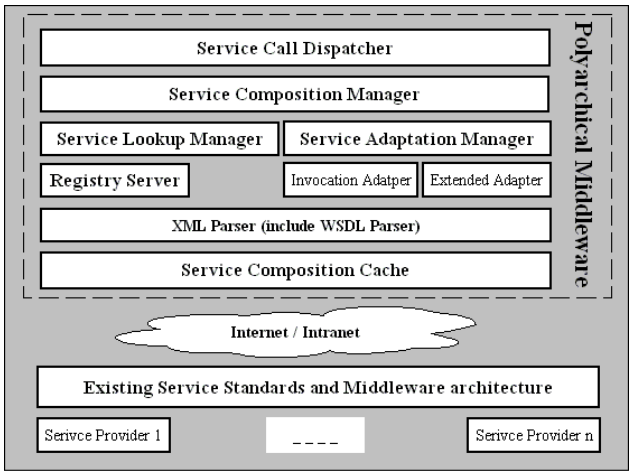
Fig. 6 illustrates the architecture of the polyarchical middleware, based on a service-oriented architecture, which sits above the existing service middleware architectures. It can assist end-users with the assembly/composition of their on-demand services at runtime, without being tied to a particular service standard and/or provider. Instead, they can discover and adapt to a given service of choice from any available resource. This framework was designed to provide support for service composition and adaptation through runtime code generation, thus abstracting service discovery and invocation details and thereby simplifying the service interoperation and integration process. Through this framework, end-users may concentrate specifically on their applications without having to worry about inconsistencies or mismatches between component standards.

- Service Call Dispatcher: provides access to this framework via Internet/Intranet. So far, a Java Tomcat HTTP server has been used to provide HTTP (TCP/IP) access to the polyarchical middleware framework. A Java Servlet container hosted by a Tomcat server is used to retrieve synchronous service invocation request

[2] Universal Description, Discovery and Integration (UDDI) is used in this case.

through URL's from end-users. In our future work, a message-based asynchronous container will also be designed to receive requests from systems that may operate in a peer-to-peer fashion.

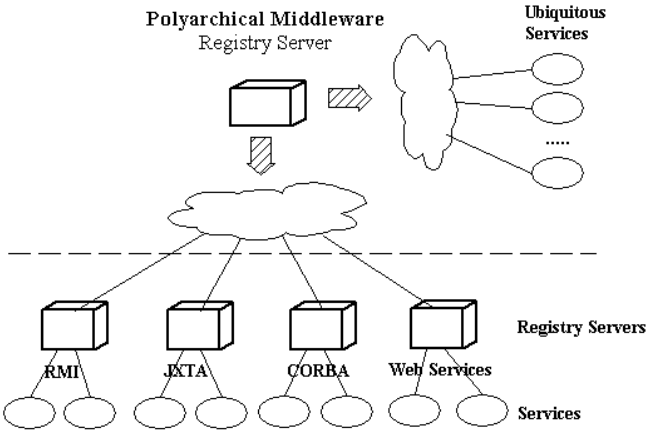
Fig. 6. Polyarchical framework structure



- Service Composition Manager: as mentioned before, is mainly responsible for runtime service composition after retrieving requests from service call dispatcher. It will initialize the virtual container for further runtime service composition. Requests of service components involved in the composition will be redirected to Service Lookup Manager for discovery/lookup appropriate components for execution. If a request arises from adaptation between two heterogeneous components, implemented using different standards, it will pass this service adaptation request to Service Adaptation Manager. In addition, any failure will also be detected and handled by the composition manager.
- Service Lookup Manager: is used to discover/lookup appropriate components for service composition execution. It uses keywords, such as service name, to search inside the local Registry Server and returns an array of matched service.
- Registry Server: is designed to support the runtime service components' discovery/lookup process. There are two types of descriptions, which may be registered/published within this Registry Server (as shown in Figure 7): These two types are: 1) *Existing Service Registry Servers:* like Java's RMI registry server, Jini's Lookup server or Web Service's registry server (usually in format of UDDI). These registry servers may be accessed/connected with the Polyarchical Middleware Registry Server through Internet/Intranet. 2) *Individual Ubiquitous Services* - descriptions about the service components individually in an ASDL format. The reason for using ASDL rather than normal WSDL has already been mentioned in Section 3.1. In Section 3.1, we considered how our ASDL can offer more information about the services to support the discovery process undertaken by the service composition manager for finding the best-fit service components. At

runtime, the service discovery/lookup process will first be executed within that local polyarchical middleware Registry Server with which the service components of interest may have registered. If no service match is found, then the service requests will be delivered to those specific registry servers individually for further discovery (i.e. the RMI, CORBA and Web Service servers). We strongly recommend that services implemented using Java RMI, or CORBA, should be manually registered within our local Registry Server in ASDL format, because the RMI and CORBA registry servers are incapable of fully supporting the service description functionality required by the ASDL. Even within the more mature Jini Lookup server [3], end-users are still required to use templates to register/lookup services.

Fig. 7. Polyarchical Middleware Registry Server



- Service Adaptation Manager: is designed to handle any service invocation adaptation requests that occurred during the composition process. At runtime, the service adaptation manager can initialize the right adapter template file to switch invocation calls from one standard to another. The adapter template file contains an invocation mechanism, which has been specified in the ASDL of each individual service component. Service invocation information for each individual component is parsed from the corresponding WSDL files referenced inside the ASDL with *ServiceDescription* tag as shown in Figure 3. At runtime, they are appended to the adapter templates files for invoking the service. At this stage, the relevant extra files, which are used to assist the service invocation, are also automatically generated. For example, the service interface may be required by a Java RMI-based service connector. In this case, the service adaptation manager will generate, at runtime, the associated RMI Interface by parsing the service description encoding using the WSDL specifications. Any failure due to changes in service components will also be detected by the polyarchical middleware at this stage and alternate components and connectors will then be sought.

- Extended Adapter: To make the polyarchical middleware more extensible, with respect to service interoperation and integration, we intend to introduce a Template Markup Language (TML) into the framework design. The TML will be used to add/describe more adapters as and when new service component standards are introduced in the future.
- Existing Middleware Standards and Service Providers: refer to the middleware technologies that provide normal middleware discovery/lookup and invocation protocols. Through the polyarchical middleware, service providers' components, developed using these middleware standards, may be integrated and interoperated by end-users.

6. Illustrative Example

In order to demonstrate the idea, an example of wireless service applications composition is used in this paper. We argue that current wireless devices can't fully utilize the network resources due to their own limitations such as CPU performance and low memory. As a result, the proposed polyarchical middleware could be used to take over the runtime more *heavier duties* (e.g. ad-hoc service discovery, assembly using virtual containers, invocation and adaptation) from wireless devices (i.e. PDA or mobile sets). In order to illustrate the ideas above, we describe the following scenario:

"*A wireless device end-user is on his way back from work. Before he arrives at home, he wants to several of his home applications/devices to be set up ahead such as: set up the timer, turn on the lamp, switch on the Kettle and turn the heating to a certain temperature. We assume that these networked home applications/devices have been purchased from a variety of different vendors, with different standards/drivers to make them work. Fortunately, the proposed polyarchical middleware comes to the rescue by receiving on-demand requests from the end-user, composing the different devices and making them work together. All the details of service composition, discovery and adaptation will be hidden away from the end-user.*"

In this case, we assume all the home application devices are driven by the software to be networked. The software could be developed using different protocols such as: Java RMI, SOAP-RPC, and CORBA. Before utilizing these services, they are required to register on the local UDDI encoding in ASDL through an interface (Fig. 9).

In order to precisely describe the on-demand service components, the end-user sends an XML-based service assembly message (Fig. 8) to inform the polyarchical middleware. After retrieving this message, the service composition manager will initialize a service virtual container for subsequent composition. The service lookup manager is responsible for discovering all the required services (i.e. clock, lamp, kettle and heating) over home private network. Finally, the matched service components are added into the virtual container for composition in the sequence defined in that service assembly message.

Fig. 8. Wireless assembly applications

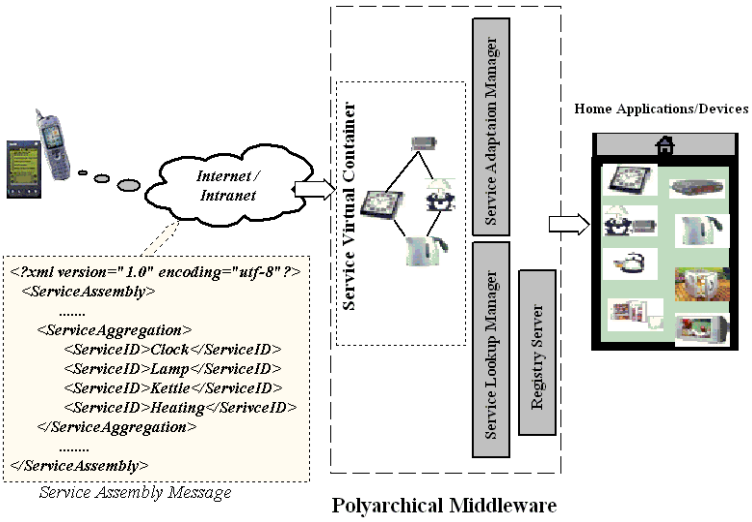
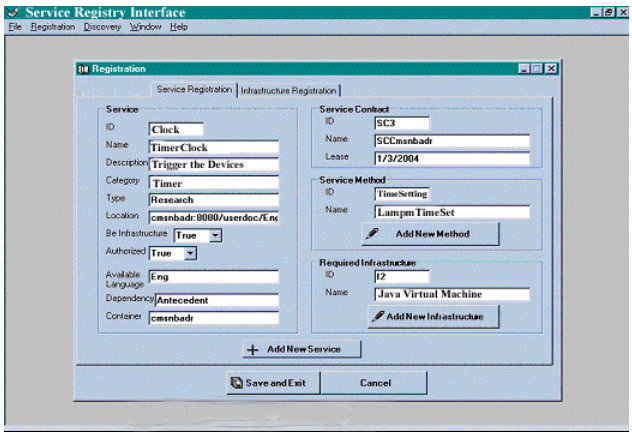


Fig. 9. Service Registry Interface



The service composition manager is also in charge of monitoring the service composition process and detecting any service failures. For instance, the *Clock* component is used to trigger the *Lamp* to switch on and the *Clock* and *Lamp* have been developed using two different standards such as Java RMI and SOAP-RPC. To allow the process to proceed, at runtime, the service adaptation manager can provide a SOAP-RPC invocation adapter (shown in following programming code excerpt) for smoothly switching the Java RMI call to a normal SOAP-RPC call. At runtime, a 'ParseWSDL.java' file is initialized by setting the parameter WSDLURL for the parser of the SOAP-RPC rpcrouter URL and serviceName(). As a result, the

polyarchical middleware maintains the service composition process without having to halt for the redesign of the component's invocation mechanism.

```
public class SoapAdapter {

public void soapConnect(String wsdlURL) throws
Exception

   {

       // Normal variable declartion

      ....

       // Runtime information filled in

       ParseWSDL parseWSDL= new ParseWSDL(wsdlURL);

       url=new URL(parseWSDL.soapURL());

       Call call=new Call();

       call.setTargetObjectURI("urn:Hello");

       call.setMethodName(parseWSDL.serviceName());

       call.setEncodingStyleURI
                (Constants.NS_URI_SOAP_ENC);

     // invoke SOAP object

      ......

   }

}
```

7. Conclusions and Future Work

In this paper we have described an approach to dynamic service composition based on polyarchical middleware that enables end-users to discover, assemble and invoke on-demand service at runtime and accomplish critical service interoperation and adaptation tasks regardless of the component standards.

Development is still underway to extend the proposed polyarchical middleware's runtime service composition abilities for interoperation and adaptation. We also

intend to extend its security mechanism and trust relationships between end-users and the proposed polyarchical middleware.

8. References

1. Corporation, M., DCOM Technical Overview, accessed date: Oct 2003, http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dndcom/html/msdn_dcomtec.asp,
2. Newmarch, J., CORBA and Jini, accessed date: April 2003, http://jan.netcomp.monash.edu.au/java/jini/tutorial/Corba.xml,
3. Newmarch, J., Jan Newmarch's Guide to JINI Technologies, Version 3.02, accessed date: Oct 2003, http://jan.netcomp.monash.edu.au/java/jini/tutorial/,
4. J.Newmarch, Web services, accessed date: Oct 2003, http://jan.netcomp.monash.edu.au/webservices/tutorial.html,
5. E.Grishikashvili, N.B., D.Reilly,M.Allen,M. Yu and A. Taleb-Bendiab;School of Computing and Mathematical Sciences Liverpool John Moores University. *Automatic Computing: A Service-Oriented Framework to Support the Developement and Management of Distributed Applications*. in *PGNET*. 2002. Liverpool John Moores University UK: School of Computing and Mathematical Sciences Liverpool John Moores University.
6. Edwards, W.K., *Core Jini (2nd Edition)*. December 28, 2000.
7. Introduction to CORBA, accessed date: May 2003, http://developer.java.sun.com/developer/onlineTraining/corba/,
8. Weiser Computer Sciences Laboratory, X.P., Ubiquitous Computing, accessed date: Oct 2003, http://www.ubiq.com/hypertext/weiser/UbiHome.html,
9. Liang-Jie Zhang, J.-Y.C., Researcher, IBM T.J. Watson Research Centre, Developing Grid Computing Application, Part1,Introduction of a Grid architecture and toolkit for building Grid solutions, accessed date: April 2003, http://www-106.ibm.com/developerworks/webservices/library/ws-grid1/,
10. M.YU, A.T.-B., D.Reilly and Wail Omar. *Multi-Standard Service Interoperation Protocol Through Polyarchical Middleware*. in *PGNET 2003*. June 2003. Liverpool John Moores University, Liverpool UK: The School of Computing and Mathematical Sciences, Liverpool John Moores University.
11. Matthew J. Duflter, N.K.M., Aleksander Slominski and Sanjiva Weerawarana;IBM T.J. Watson Research Center, Web Service Invocation Framework (WSIF), accessed date: July 2003, http://www.research.ibm.com/people/b/bth/OOWS2001/duftler.pdf,
12. Erik christensen, M.F.C., IBM Research; Greg Meredith, Microsoft;Sanjiva weerawarana, IBM Research, Web Services Description Language (WSDL) 1.1, accessed date: Oct 2003, http://www.w3.org/TR/wsdl,
13. Wade Wassenberg, S.E., Motorola ISD, Protocol Independent Programming Using Openwings Connector Services, accessed date: Oct 2003, http://www.openwings.org/download/specs/OpenwingsConnectorWhitepaper.pdf,
14. M.Mochizuki, A Middleware Architecture for the Improvised Assembly of Component-Based Applications. 2000. p. Keio University.
15. M.Mochizuki and H.TokuDa, Improvised Assembly Mechanism for Component-Based Mobile Applications. IEICE TRANS.COMMUN., 2001. **Vol.E84-B**.
16. Shin, S., Sun One: Now and Future, accessed date: Sep 2003, http://www.plurb.com/webservices/SunONENowAndFuture_V4.pdf,

A System for Distributed Enactment of Composite Web Services

S.J.Woodman[1], D.J.Palmer[1], S.K.Shrivastava[1], S.M.Wheater[2]

[1]School of Computing Science, University of Newcastle, Newcastle upon Tyne, UK
{S.J.Woodman, Doug.Palmer, Santosh.Shrivastava}@ncl.ac.uk
[2]Arjuna Technologies Limited, Nanotechnology Centre, Newcastle upon Tyne, NE1 7RU, UK Stuart.Wheater@arjuna.com

Abstract. Availability of a wide variety of Web services over the Internet offers opportunities of providing new services built by composing them out of existing ones. Service composition poses a number of challenges. A composite service can be very complex in structure, containing many temporal and data-flow dependencies between their constituent services. However, constituent service operations must be scheduled to run respecting these dependencies, despite the possibility of intervening processor and network failures. The architecture must be scalable, providing a decentralised coordination of service execution rather than based on a centralised scheduler; this is particularly important for services spanning different organisations, where reliance on centralised coordination would be impractical. This paper presents the design and implementation of DECS: a workflow management system for Distributed Enactment of Composite Services. A novel feature of DECS is the separation between specification of service composition and its enactment. A DECS service specification can be deployed either for centralised or decentralised coordination, depending upon inter-organisational requirements. A prototype implementation of DECS has been performed using J2EE middleware. The paper describes the DECS task model for specifying service composition and the middleware services that have been implemented in J2EE for coordinating service execution.

1 Introduction

It is becoming increasingly common for a Web service to make use of Web services offered by different organisations. We term such an inter-organisational service a "Composite Web Services" (CSs). There is much research interest in developing high-level tools for CS creation and management, including service specification languages and run time environments for coordinating the execution of (enactment of) constituent services. This paper presents the design and implementation of DECS: a workflow management system for Distributed Enactment of Composite Services. There are several features DECS that make it novel and distinct from existing workflow management systems for Web service enactment.

1. *Flexible Coordination:* workflow management systems for Web service enactment specify the composition of a CS as a business process. Such a specification should be sufficiently abstract, uncluttered with specific details of enactment. Specifying the enactment of the business process is a concrete operation which requires further information that is critically dependent on organisational issues. For example, the organisations involved in a CS may have a peer-to-peer business relationship, in which case, a decentralised enactment seems a natural choice, with each organisation responsible for its part of the process. Whereas in a hierarchic relationship, a centralised enactment may well be deemed more appropriate. DECS supports the separation between the specification of service composition and its enactment, enabling the organisations deploying the service to decide how they wish to coordinate it, rather than the designer of the business process.
   DECS provides the option of both centralised coordination as shown in Figure 1 and decentralised coordination, as shown in Figure 2. Should the coordination of the service be spread across multiple servers as in Figure 2 a higher level of fault tolerance is provided. In such cases, each server makes the invocations of the constituent web services for its part of the CS, and communicates via a coordination protocol with its peers to orchestrate the overall execution. Should a coordinating server fail or leave, it is possible to move the CSs which that server was coordinating to another server. The cost of doing so is proportionate to the number of CSs being coordinated and the complexity of those CSs.

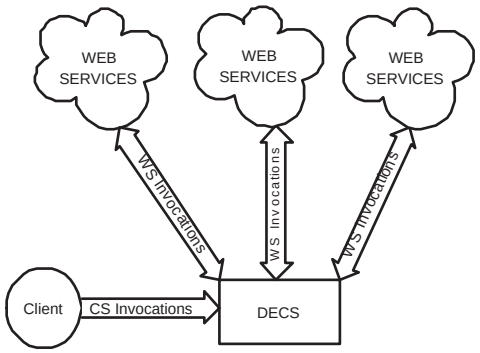


Fig. 1. Centralised Coordination of a Composite Service

2. *Support for reconfiguration:* It is expected that the execution of a CS could take a long time to complete, of the order of days or weeks, and may contain long periods of inactivity often due to the constituent applications requiring

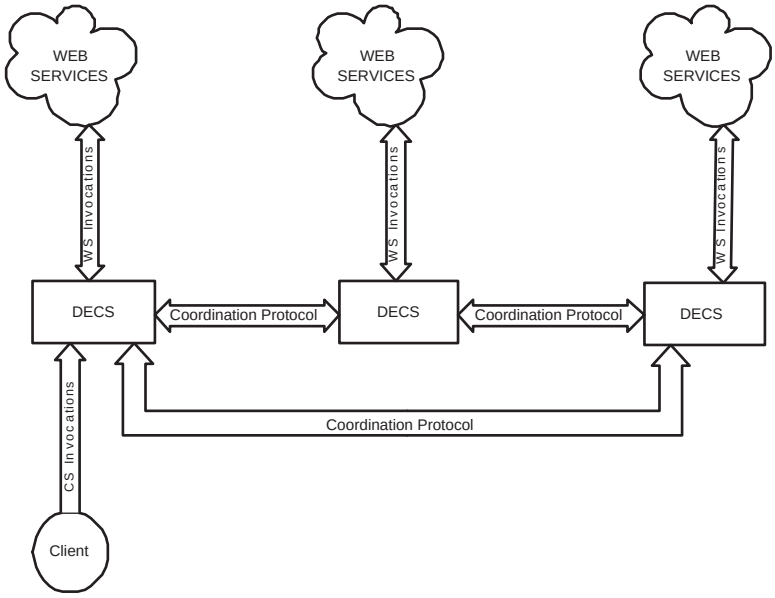


Fig. 2. Decentralised Coordination of a Composite Service

user inputs. It should be possible therefore to reconfigure a CS dynamically because, for example, services may be moved, machines may fail or users requirements may change. DECS provides basic system support for reconfiguration (see section 22).

3. *Support for preserving organisational Autonomy:* A further advantage of distributed coordination is that it enables organisations to collaborate whilst maintaining autonomy. When an inter-organisation business process is created, it can be deployed such that each organisation coordinates the parts which they are responsible for and which act upon their internal data. Each organisation is aware of the data relating to those tasks which it is coordinating, and also certain pieces of data which they have requested from the other organisations.

The remainder of this paper is structured as follows: section 2 gives an overview of the system; section 3 describes the task model of DECS; in section 4 we describe the system architecture and its implementation; section 5 we discuss patterns for distributing coordination; section 6 describes related work, finally section 7 concludes the paper.

2 System Overview

We discuss how our system has been designed to meet the application requirements implied above, namely: scalability, dependability, interoperability, dy-

namic reconfiguration and flexible task composition. The design of the system has been influenced by our earlier work on the OPENflow distributed workflow management system [1,2].

- Scalability: Once a composite service is deployed, there is no reliance on any centralised service which could limit scalability. The decoupling of the business process definition from the deployment aids scalability too it is possible to reconfigure the business process at run-time if more resources are needed.
- Flexible Task Composition: The system supports a simple yet powerful task model (see section 3). This allows the composition of complex services from simple services located both on the local machine and remote web services. A task can perform application specific input selection (e.g. obtain input from one of several sources) and terminate in one of several outcomes.
- Dependability: Dependability is implemented at two levels in DECS: application level and system level. Due to the flexible task composition model mentioned above it is possible to specify different tasks and processes to handle a variety of exceptional circumstances, for instance, compensating tasks, alternative tasks etc. At the system level, DECS makes use of the facilities provided by modern component middleware (J2EE) to ensure that all information relating to the composite service: its structure, data and state are persistent. All interactions between different modules of DECS make use of transactions to ensure that the data remains consistent and tasks do not interfere with each other. If a coordinating node fails mid process, it is possible to recreate the state of the CS and continue processing from where the failure occurred. This is discussed further in Section 4.6.
- Interoperability: The system has been designed to run in any J2EE compliant application server, thereby supporting system level interoperability. As DECS can invoke web services with arbitrary interfaces, application level interoperability is provided.
- Dynamic reconfiguration: The task model referred to earlier is expressive enough to represent dependencies (dataflow and notification) between tasks. The execution environment exposes low level operations for making changes to the structure of the CS by altering the tasks and the dependencies between them. We are currently working on making this dynamic reconfiguration support transactional to ensure that changes are carried out in a consistent manner despite concurrent reconfigurations and machine failures. This work is being done along the lines of our earlier implementation [2].

3 Task Model

DECS makes use of a flexible task model to describe the structure of Composite Services. The schema for such a task model must be expressive enough to allow arbitrary CSs to be defined. Our schema is defined in terms of tasks, temporal dependencies and data dependencies. A task represents an application specific

unit of work that requires specified input data and produces specified output data and corresponds to an invocation of a web service. A task instance is modelled as receiving one input message, and sending multiple output messages. The web service invoked is treated as a black box with the input and output data specified by the WSDL document associated with the service [3].

A temporal dependency represents the situation where a down-stream task cannot start until an up-stream task has terminated in a particular state. For example, a goods dispatch task should not be invoked until a payment-capture task has completed successfully. A data dependency indicates that a down-stream task requires input data from the up-stream task. Each data dependency implicitly has a temporal dependency associated with it. For example, a dispatch task requires shipping-address which is part of the output from order task. Implicitly this is not available until the order task has completed [1].

Some of the salient points of the task model which aid flexible composition of CSs are presented below:

- Alternative input sources: Any part of the input data can be acquired from more than one source. This enables the introduction of redundant data sources providing application level fault tolerance. The input data for a task is described at the granularity level of wsdl:part. Support for finer granularity data than wsdl:part is described in Section 7.
- Alternative outputs: A task can terminate in one of several states producing distinct outcomes. In terms of WSDL, one of these states will correspond to an "output" and all others to a "fault". This allows different down-stream tasks to be executed depending on the outcome of an up-stream task. For example, compensation in the event of a fault.
- Compound Tasks: A task can be composed from other tasks. The composed tasks themselves can be simple tasks or compound tasks. This is the principle way of providing abstraction.
- Genesis tasks: A genesis task represents a placeholder for a task structure and is used for on-demand instantiation. This allows the system to only instantiate the parts of a complex task which are necessary, and also allows execution of repetitive tasks.

Each input and output message can contain several parts, each representing a piece of data which can be requested by other tasks. Task t2 in Figure 3 has one input message containing two parts, namely i1 and i2. There are two output messages for task t2, O1 and O2, each containing one part, o1 and o2 respectively. A task begins its life in the wait state, awaiting its input data to be complete. When the input message is complete (i.e., all the data dependency and temporal dependencies are satisfied) the task enters the ready state. If alternative inputs become available simultaneously the one with the highest priority is used. Note that the source of an input part can be from an output message (e.g. d1), or an input message (e.g. d2), the latter represents the case when an input is consumed by more than one task. Temporal dependencies are depicted as a dotted line, for example n1 and data dependencies are shown as solid lines. A compound task

such as t1 can contain multiple output messages, with each part having several possible sources. The task will terminate when one output message is complete. If multiple output messages become available simultaneously, the one with the highest priority will be chosen. In section 6 on related work, we compare our task model with the recent proposal for Web service enactment, BPEL [4].

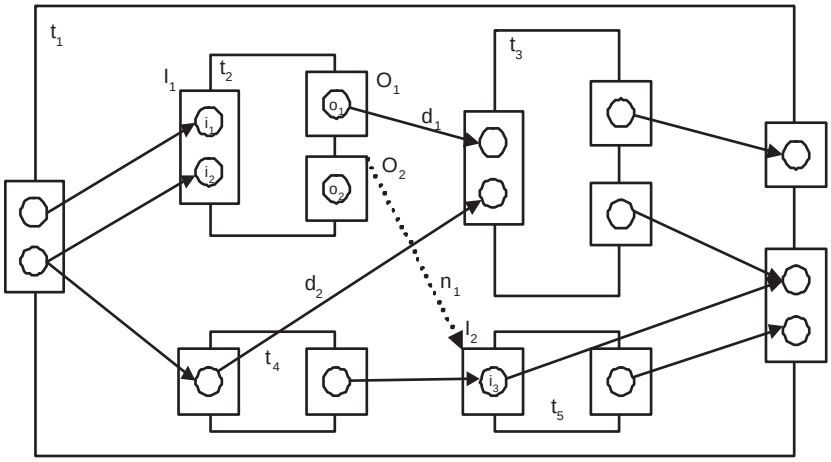


Fig. 3. A Compound Process

4 System Design and Implementation

The DECS system has been designed and implemented using J2EE middleware. The current version has been tested to run within the JBoss application server [5].

4.1 Overall structure

The structure of the system is shown in Figure 4. The figure is intended to show how a clients request to execute CS enters the Process Initiator; an instance of the process definition is taken from the Process Definition Repository (PDR) and added to the Process Instance Repository (PIR). The coordinator then uses this data to invoke the web services which compose the CS and inform other coordinating servers of data in which they have registered an interest. Every client request instantiates a new and unique process definition instance. The J2EE technologies used to implement the different modules are shown in Figure 5 which matches the structure of Figure 4.

Using the J2EE environment allows DECS to make use of the rich set of functionality which J2EE application servers provide. For example, uniform access to persistent storage, flexible transaction control, a unified security model. DECS utilises the Entity Beans, Session Beans and Message Driven Beans from the J2EE architecture. An Entity Bean represents a business object that exists in the enterprise application. The application server controls access to Entity Beans, guaranteeing atomicity, consistency, isolation and durability. A Session Bean provides the business logic for a J2EE application. Clients call methods of the Session Bean to interact with the application. Message Driven Beans provide asynchronous behaviour by acting as message listeners for the Java Message Service (JMS). JMS facilitates the exchange of messages among software applications over a network. The prototype of DECS is interoperable in that it will run in any J2EE application server as no proprietary extensions have been exploited.

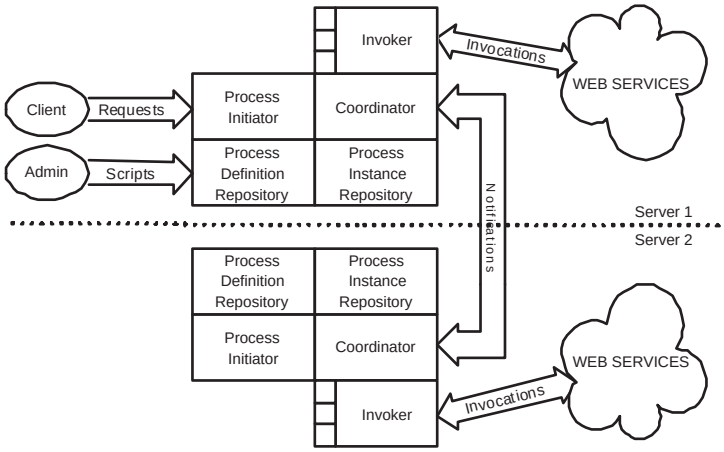


Fig. 4. System Architecture

4.2 Process Initiator

The process initiator is able to dynamically deploy web service endpoints which can be used by a client to invoke the web service which corresponds to a process. When a composite service is deployed in the system, an endpoint is generated which allows a client to invoke it. This enables us to achieve transparency from the client's perspective, whereby the client may be unaware that they are invoking a web service which is implemented as a process. This style differs from many coordination engines which expose one interface to invoke many processes. For example, web service clients of the CARNOT workflow engine must send a message to the WsWorkflowSessionService" endpoint passing the name of the

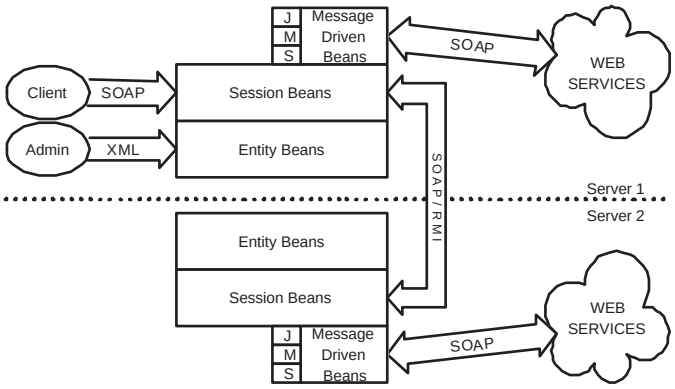


Fig. 5. System Technologies

process to invoke and a context. One of the problems with this style of interaction is that there is no standard way of locating the required service. Processes which can be invoked using the Process Initiator can be advertised using any current advertising method, e.g. UDDI [6].

When designing a CS, one server is designated as the root controller for that CS. This is usually the server which will coordinate the first task in the CS but this is not a requirement. The root controller is the only server on which the Process Initiator deploys an endpoint. When a CS is invoked by a client, an instance of the process definition is created in the PIR based on the process definition stored in the PDR. This contains all the data necessary to run the CS, such as the tasks involved and the inter-task dependencies which must be satisfied. The root controller also sends a message to the other servers coordinating the execution requesting that they create an instance of the fragment of the process definition that they are responsible for. This is discussed further in Section 5. The initial values received in the client request are then added to the "root" task in the repository and those tasks whose input dependencies are satisfied are invoked.

4.3 Process Definition Repository (PDR)

The Process Definition Repository stores process definitions and provides a method for instantiating an instance of a process. DECS provides tool support adding an XML based process definition script to the PDR. This can either be a complete process definition for centralised coordination or a partial definition if decentralised coordination is required. The PDR is implemented as a collection of Java Entity Beans deployed in the J2EE application server.

A process definition is represented by a schema which matches the task model described in the previous section, in terms of tasks and dependencies. We have

made use of the concepts of a scripting language developed for a previous distributed workflow system and adapted it to suit web services style invocations. The scripting language was designed to express composite service composition and inter-task dependencies of fault tolerant distributed applications whose executions could span arbitrary large durations [7].

4.4 Process Instance Repository (PIR)

The system will instantiate a process instance based on a process definition for each request from a client. The data related to each process instance is stored in the PIR which is also implemented as a collection of Entity Beans The state of the process instance is persistent and each change is made persistent, this means that coordination of process instances can be continued after machine failure.

Creating a process instance from a process definition involves three steps:

- Create the local structure of the process instance in the PIR according to the definition stored in the PDR.
- Instantiate the fragments of the schema which reside on remote nodes. This is described further in Section 6.
- Insert the initial values into the process instance from the client request.

4.5 Coordinator

The coordinator in DECS orchestrates the execution of the CS across multiple nodes. This involves checking for input availability, maintaining state of the task and propagating the results both locally and remotely. The prototype uses Session Beans to implement the business logic associated with coordination.

When a process definition is instantiated all the tasks are in the wait state. As input data is added to the tasks they are checked to see if their input message is complete. Once the input message is complete the task moves to a ready state and is put on a persistent JMS queue to indicate that the task is invocable. At this stage, the coordinator checks to see if there are any input dependencies or notifications on the task's input message. If local dependencies (either data or temporal) exist, the data is propagated locally to these tasks. If a dependency is remote, this initiates a notification of the data to the remote coordinator. These are shown in Figure 8(iii) as d1 and n1 respectively. When propagating data to the remote coordinator, either SOAP or Java RMI can be used. SOAP is intended to be used as the primary communication method but Java RMI can be used to optimise the communications if both coordinators are located on the same network. In both cases, the local coordinator must communicate with the remote coordinator via an RPC style call. The parameters of the notification include the unique identifier of the process instance which is the source of the data and the data value. The action of inserting the data part to the remote task has the side effect of checking the task to see if it can be executed. The semantics of adding any data, local or remote, to a task results in the task being executed as soon as its input message is available. When the output message of

the root task is complete, the Process Initiator will create a SOAP response and send it back to the client.

4.6 Invoker

The invoker is responsible for invoking the web services which comprise the composite service. The invoker is implemented as Message Driven Beans in the application server. This allows us model the asynchronous behaviour that is required to invoke the constituent web services when their dependencies have been fulfilled.

To cope with the consequences of coordinator failure, the designer of the service is currently able to specify one of two actions to be taken on restart for each task which was executing when the failure occurred. These correspond to whether they wish the constituent web services to be invoked at most once or at least once. If the designer specifies they wish to use at most once semantics, transactions are not used in the invoker. The invoker is not aware that an instance of this service was executing when it failed, so it does not attempt to re-invoke the web service request. If at least once semantics are required, everything performed by the invoker is within one transaction. This includes obtaining the input data for the web service, invoking that service, receiving the results and adding them to the PIR. If the invoker fails at any point within this transaction, it will be rolled back on restart. This results in the web service being invoked with the same input data again, once the system has recovered. If an end-to-end transaction protocol were available, it would be possible to achieve exactly once execution semantics for the constituent web services of the process. This is because any failure in the coordinator would be propagated to the constituent web services which could rollback their execution too. On restart, the invoker would send the SOAP request again and eventually a successful execution would occur.

5 Distribution Patterns

It is the responsibility of the designer of the service to specify how they wish to distribute the coordination of the service. Simple CSs may be coordinated centrally, as shown in Figure 6. This is the simplest scenario, where a central node performs all the invocations of the services necessary to complete the CS. More complex scenarios can be built, where the coordination of the CS is divided between multiple nodes which run DECS. One example is given in Figure 7 where the service is divided between two nodes. How to divide the service is an application specific decision. If we consider the notion of a Virtual Enterprise where a set of companies wish to collaborate to provide a service, the division of coordination could be along organisational boundaries. This allows each organisation to coordinate their own part of the service, and possibly utilise their private services. For example, company S wishes to sell a product and use company D to deliver the product. The composite service could be divided such that nodes at S coordinate the ordering of the product and payment and then the nodes at D

coordinate scheduling of the delivery of the product. A division such as this has some advantages: firstly, S can integrate the order with their own procurement process allowing re-ordering of stock if necessary and D can integrate with their private delivery scheduling services; secondly, data security is higher as only the minimum amount of data is passed across from one organisation to the other to allow the CS to continue its execution.

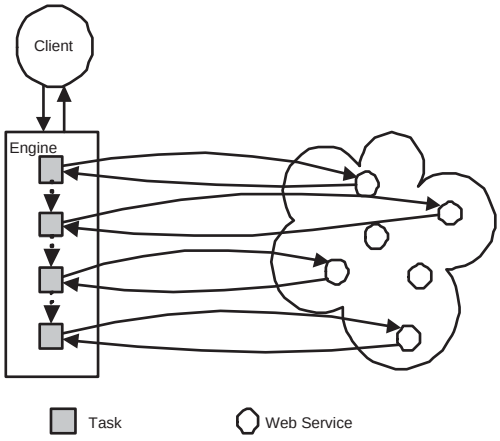


Fig. 6. Centralised Coordination

The system aims to give each node the minimum information necessary to coordinate the execution of that part of the CS. Each node only stores the data about the tasks which it is coordinating, the internal dependencies which must be satisfied and the external notifications which must be sent and received. The node is not aware of what the other nodes are doing, or what tasks they are coordinating. This is intended to provide autonomy; it makes it attractive to businesses that do not wish to disclose all of their internals but do wish to integrate their business processes with a trading partner.

Figure 8 shows an example of how a very simplistic CS, A could be divided to run over three servers, X, Y and Z. Figure 8(i) shows the overall CS with figures 8(ii) to 8(iv) showing the three servers views of the service. Server X is delegated as the controller of the CS so it is this server which exposes an endpoint allowing the service to be invoked. When a client request is received at X for service A, the first task to be executed will be B. When B completes, there are no dependent tasks at server X, but there is a notification request for the results to be sent to Y. When Y receives the results of B via a notification, task C is executed (by Y) and on completion the results sent via a notification to Z. As there is also a dependency for task E at Y, the data is propagated locally to task Es input message. As the input message for task E is not yet complete, the task is not invoker. At server Z, task D's input message is complete by receiving

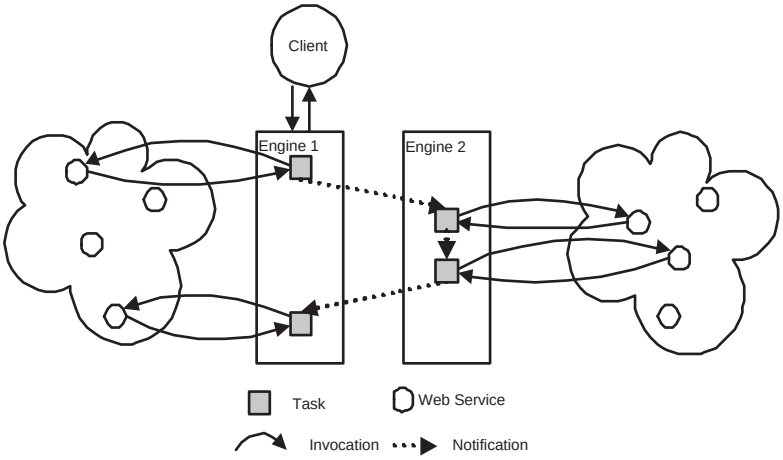


Fig. 7. Decentralised Coordination

the notification from Y so task D is executed. A notification is sent from Z to Y on completion of D which causes the input message of task E to be complete and thus task E fired. Completion of task E causes a notification to be sent to X with the results which are used as input for task F. On completion of task F, the output message for the compound task A is complete, so the result is sent to the client.

6 Related Work

There are several industry led efforts aimed at developing (often competing!) standards for specifying, composing and coordinating the execution of CSs. The specifications are still evolving and often ill defined [8]. We compare our task model with a recently proposed Web service execution language standard. The aim of Business Process Execution Language for Web Services (BPEL) [4] is to provide a standard for specifying business process behaviour and business process interactions, for applications composed from Web services. We are going to focus on a comparison of the approach taken by BPEL for specifying business process behaviour and that taken by DECS.

Like DECS the BPEL process model allows business partners interact through peer-level conversations, using both synchronous and asynchronous messages. These conversations are carried out between the partners using specified sets of Web services. BPEL has been designed to allow the coordination of distributed web services in a centralised manner. The coordination itself was not designed to be distributed. The coordination model is tailored towards implementations that have a centralised state upon which a rich set of activities can act. The result is that providing a decentralised enactment engine is difficult especially if

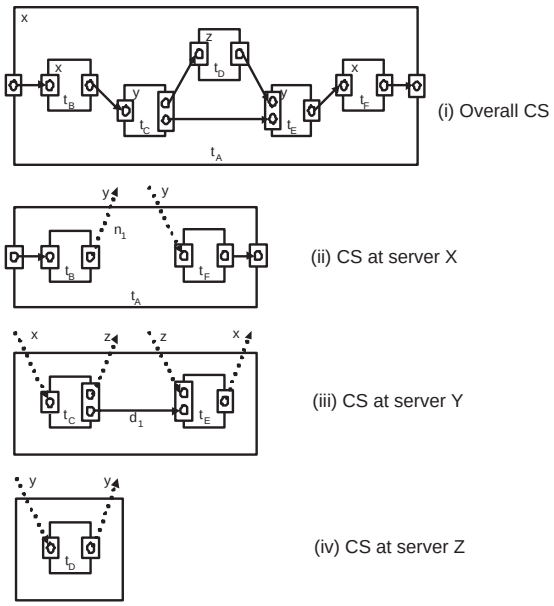


Fig. 8. Division of a CS

the intention is to deploy the workflow in a wide area network. In comparison, the DECS task model was designed such that each task holds a small amount of state and there is a minimal set of services which are able to act on this state. This results in a model which is easier to distribute as each task encapsulates its own state and it is easier to migrate task coordination to another node.

BPEL relies on specifying a large set of explicit control flow activities such as forking, joining and conditionals. Conversely, control flow in DECS is implicit and specified through dependencies. These constructs have been found to be sufficient to specify complex processes. Another consequence of BPEL having such an extensive set of features which can be combined to specify a process is that the resulting process cannot be easily analyzed to verify such properties as eventual termination. The DECS model is much simpler and amenable to analysis, allowing both temporal and correctness properties to be checked [9].

There are a number of systems available which coordinate the orchestration of composite services produced both in the academic community and in industry. Some of these are discussed below and briefly compared and contrasted to DECS.

Collaxa [10] is well-known product that can enact composite services specified in BPEL. It is a centralised coordination engine; this is inevitable, given the observations on BPEL above. As such, it cannot offer the same level of flexibility in deployment scenarios, dependability and scalability that DECS intends to provide.

Both eFlow [11] and BioOpera [12] explore the declarative composition of services but concentrate on a centralised orchestration model. Successful efforts have been made in eFlow to allow dynamic refactoring of services although this is simplified due to a central, global view being available.

DECS has been designed to run in a J2EE environment to allow portability and ease of integration into existing enterprise applications. DySCo [13] offers many similar features to DECS, such as decentralised coordination but portability has not been addressed to the same level: it is not designed to run in standard middleware such as J2EE. Conversely, CARNOT [14] offers portability through a J2EE implementation, but does not support distributed orchestration of composite services.

Another approach which has been explored is that of SELF-SERV [15], based on a declarative state-chart oriented language. It also includes a peer-to-peer coordination model and provides support for equivalent services through the use of service communities. It comes closest to DECS in terms of design aims and functionality.

7 Concluding Remarks

We have presented the design and implementation of DECS: a workflow management system for Distributed Enactment of Composite Services. A novel feature of DECS is the separation between specification of service composition and its enactment. A DECS service specification can be deployed either for centralised or decentralised coordination, depending upon inter-organisational requirements. A prototype implementation of DECS has been performed using J2EE middleware.

A suite of common services is being developed as part of the DECS. Such services include:

- User input service: it is likely that some CSs will require input from users at different parts of the execution. For this reason we are developing a servlet based user interface which will allow users to input parameters to be used in the execution. The data entered may be used to determine the consequent flow of execution or to provide advanced error recovery.
- Send and Receive services: In order to allow asynchronous communications services will be developed which will send or receive a message. Tasks which utilise these services can be added to any CS, thus potentially providing a fully asynchronous CS. We envisage more web services becoming available which require message based communications rather than the RPC style services which are common at present. Such document exchange web services are more versatile and allow easier integration of business processes. However, with the introduction of this style of interaction problems are introduced such as message correlation and temporal issues. These will be investigated further.
- Administrative Services: services will be provided which allow a user (with appropriate permissions) to deploy, remove and dynamically reconfigure pro-

cess definitions. Care must be taken when refactoring a service which is distributed across multiple nodes to ensure that deadlock is prevented. This could occur in cases where tasks are removed upon which a remote task is awaiting a notification. The service will provide mechanisms to ensure that this situation does not occur.

– Transformation of complex types: At present, the system is able to manipulate the flow of data at the granularity of WSDL parts. However, if a complex type is defined in WSDL the system treats this as a black box and cannot address internal fields. We see this as a deficiency in the system so aim to provide a service which is able to transform complex types so that they conform to another schema. This is likely to be done using XSLT, with the designer of the CS providing a style sheet describing the transformation required. An example where this would be useful would be extracting the invoice-number from an invoice type that was returned from the order service and use it as the input to another service.

Acknowledgements

Discussions with Gustavo Alonso clarified our ideas. This work is part-funded by the UK EPSRC under grant GR/N35953/01: Information Co-ordination and Sharing in Virtual Environments; by the European Union under Project IST-2001-37126: ADAPT (Middleware Technologies for Adaptive and Composable Distributed Components; and by the UK DTI e-Science programme under project "GridMist: Middleware Services and Tools for Managing Resource Sharing in Virtual Organisations". This work will be continued in the EPSRC funded project GR/S63199: "Trusted Coordination in Dynamic Virtual Organisations".

References

[1] S. M. Wheater, S. K. Shrivastava and F. Ranno A CORBA Compliant Transactional Workflow System for Internet Applications, Proc. Of IFIP Intl. Conference on Distributed Systems Platforms and Open Distributed Processing, Middleware 98, (N. Davies, K. Raymond, J. Seitz, eds.), Springer-Verlag, London, 1998, ISBN 1-85233-088-0, pp. 3-18.

[2] J. J. Halliday, S. K. Shrivastava and S. M. Wheater, Flexible Workflow Management in the OPENflow system, Proc. of 5th IEEE/OMG International Enterprise Distributed Object Computing Conference (EDOC 2001), September 2001, Seattle, pp. 82-92.

[3] Web Services Description Language (WSDL) 1.1.http://www.w3.org/TR/wsdl.html as viewed July 2003

[4] Business Process Execution Language for Web Services (BPEL4WS) version 1.1. http://www.w3.org/TR/2002/NOTE-wscl10-20020314/ as viewed July 2003

[5] JBoss application server: www.jboss.org

[6] UDDI Specification, OASIS http://uddi.org/

[7] Ranno, F., Shrivastava, S.K., and Wheater, S.M., A Language for Specifying the Composition of Reliable Distributed Applications, 18th IEEE Intl. Conf. on Distributed Computing Systems, ICDCS98, Amsterdam, May 1998, pp. 534-543.
[8] W.M.P. van der Aalst, Dont go with the flow: Web services composition standards exposed, IEEE Intelligent Systems, Jan/Feb 2003.
[9] C. Karamanolis, D. Giannakopoulou, J. Magee and S.M. Wheater, Model Checking of Workflow Schemas, Proc. of 4th IEEE/OMG International Enterprise Distributed Object Computing Conference (EDOC 2000), September 2000, Makuhari, Japan.
[10] Collaxa: BPEL Orchestration Engine. http://www.collaxa.com as viewed July 2003
[11] Casati, F., Ilnicki, S., Jin, L., Shan, M., An Open, Flexible, and Configurable System for E-Service Composition, Second International Workshop on Advance Issues of E-Commerce and Web-Based Information Systems. 2000. Milpitas, California.
[12] Bausch, W., Pautasso, C., Schaeppi, R., and Alonso, G, BioOpera: Cluster-aware Computting, 4th IEEE International Conference on Cluster Computing. Chicago, USA.
[13] Piccinelli, G., Finkelstein, A., Williams, S.L., Service-Oriented Workflows: The DySCO Framework, Proceedings of Euromicro Conference, Antalya, Turkey, 2003
[14] CARNOT Workflow Engine. http://www.carnot.ag/en/ as viewed July 2003
[15] Benatallah, B., Sheng, Q.Z., and Dumas, M., The Self-Serv Environment for Web Services Composition, IEEE Internet Computing, 2003. 7(1): p. 40-48.

Service Oriented Computing in the context of Mathematical Software

Yannis Chicha and Marc Gaëtano
chicha@essi.fr, gaetano@essi.fr

I3S/Université de Nice-Sophia Antipolis
ESSI, 930 route des Colles, BP 145
06903 Sophia Antipolis Cedex, France
tel. : +33 (0)492 965 157
fax.: +33 (0)492 965 055

Abstract. This paper explores the possible migration of mathematical packages to autonomous, platform-independent web services. Mathematical computing has been a major branch of computer science since the early ages. Both numerical and symbolic computation brought up many highly specialized and efficient pieces of software.
Such software can be used to solve a broad variety of mathematical problems whose applications cover various domains. Unfortunately, most of these mathematical packages remain unknown from a significant part of their potential users: too difficult to use for non-specialists, not available on the user's platform or environment, or simply not advertised enough, mathematical packages rarely evolve beyond the stage of prototypes. A natural way to improve the accessibility of these packages is to turn them into mathematical web services.
After discussing the problem-oriented nature of mathematical packages, we present some results in mathematical service discovery. We introduce examples of ontologies and taxonomies dedicated to mathematical problem description. Finally, we show the importance of Service Brokering and Planning in this context and envision how mathematical services could be exploited in the context of web-based e-business.

1 Introduction

Mathematical computing has been a major branch of computer science since the early ages. Both numerical and symbolic computation brought up many highly specialized and efficient pieces of software. This paper explores the possible migration of conventional mathematical packages to autonomous, platform-independent mathematical web services, enabling users to access agile mathematical processes that can adapt and respond to high level queries.

Mathematical software can be used to solve a broad variety of problems whose applications cover various domains such as engineering, medicine, finance, management, education or even art. Unfortunately, most of these mathematical packages remain unknown from a significant part of their potential users: too

difficult to use for non-specialists, not available on the user's platform or environment, or simply not advertised enough, mathematical packages rarely evolve beyond the stage of prototypes. A natural way to improve the accessibility of these packages is to turn them into mathematical web services, providing:

- automated service discovery based on semantics.
- uniform access to routines.
- composition of services to solve complex problems.

After discussing the problem-oriented nature of mathematical packages and how this relates to a classical service-oriented design, we present some of the current results and practices in mathematical service discovery. We also introduce examples of ontologies and taxonomies dedicated to mathematical problem description. Service brokering and composition planning in the context of mathematics are crucial because it is often the case that the same mathematical problem can be solved by many different packages in many different ways. We discuss these challenging issues and how to use mathematical problem solving planners. We then envision how mathematical services could be exploited in the context of web-based e-business. Finally, before a few concluding remarks, we present a case study carried out within a European Union funded project called MONET [1], which aims at demonstrating the application of the latest ideas and technologies for creating a semantic web applied to the world of mathematical software.

2 Service-oriented and Problem-oriented computing

The primary application for web services in the world of mathematical software is to simplify point-to-point integration between systems, thereby allowing the use of one package from within another. Typical examples include the use of a highly specialized package from a general purpose computer algebra system like Maple [2] or Mathematica [3]. For example, [4] describes the use of Bernina functions from Maple. This application, however, only scratches the surface of the true potential of mathematical web services. Service oriented computing can enable users to access agile mathematical processes that can adapt and respond to high level queries, through the use of loosely coupled, standards-based mathematical services. Unlike general web services, mathematical services can be organized according to the problem they are able to solve. These problems themselves can be organized as a tree structure reflecting the natural inheritance between mathematical problems. For example, Figure 1 shows a sub-classification for equation solving problems.

Using suitable taxonomies and ontologies, problems solved by mathematical packages can be unambiguously described and can act as the search keys for finding mathematical services. Currently, there are only few examples of such taxonomies (e.g. GAMS [5]) and these have to be developed. In a problem-oriented environment, the entry point for a computation is the problem to solve and not the package used to solve it.

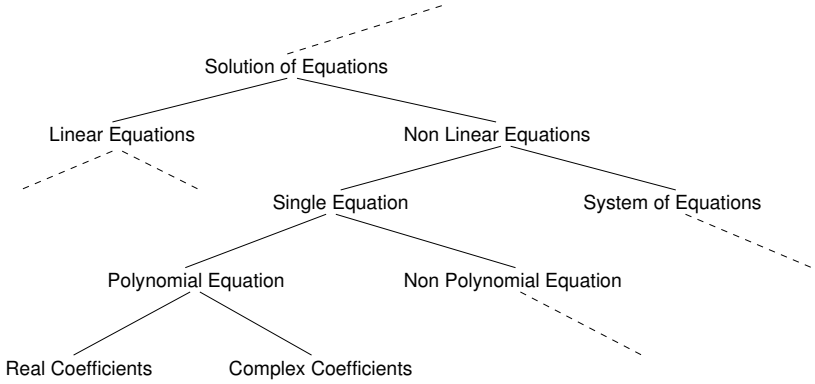


Fig. 1. Problems hierarchy

Figure 2 shows two problems ("solving a differential equation" and "computing a groebner basis") possibly solved by a few different packages (using `dsolve` and `groebner` functions). The discovery of a package solving a particular problem can be done using a special service, which takes as input the description of a problem and returns a list of matching packages. For example, "computing a groebner basis" can be done by invoking the "groebner" function of the `Maple`, `Singular`, or `Gb` packages.

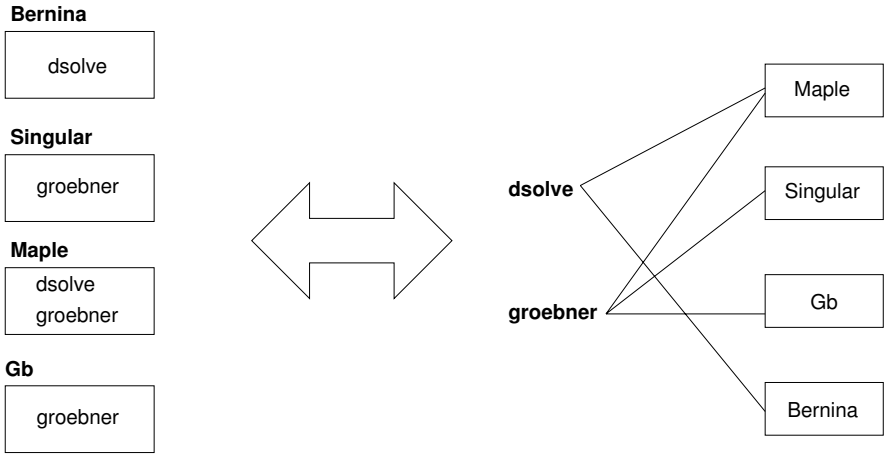


Fig. 2. Problem-oriented vs Service-oriented

3 Semantics and Service discovery

Many mathematical applications require precise information on the problem to solve and data provided. Semantics is useful in this context and essential for service discovery. Fine details, such as the degree of polynomials involved in a computation, may make the difference when choosing what service to invoke.

The web services community provides a way to register services to a central entity to help clients locate appropriate services when needed. The UDDI [6] standard allows one to describe a service and to provide enough information for a client to connect to this service. However, a recent study [7] argues that UDDI – in the context of mathematical web services – does not provide enough support for precise semantics, and thus can not be used to locate mathematical services. A "Mathematical Service Matcher" (MSM) should locate services according to the precise descriptions of the problems they solve. Also, it should be possible for a MSM to advise clients about services when no exact match is found. Mathematical problems are not independent from one another and relationships exist. One problem A may generalize to another problem B. In this case, if a service advertises B but not A, the MSM should be able to tell the client that no exact match was found but the service doing B could be suitable.

Support for these sophisticated search functions are typically provided by specialized taxonomies, necessary to allow semantic matching for service discovery. This is what OpenMath [8] and Content MathML [9] strive to provide. OpenMath is a standard for representing mathematical objects with their semantics, allowing them to be exchanged between computer programs, stored in databases, or published on the worldwide web. MathML is a language, based on XML and accepted as a W3C recommendation, to represent and manipulate mathematical objects on the worldwide web just as HTML did for simple text. MathML Presentation defines a language for presentation in a browser while Content MathML defines a language for semantics of presented mathematical objects. Unfortunately, these languages do not describe actions, only objects. To our knowledge, the Guide to Available Mathematical Services (GAMS) provides the only existing classification to describe such actions. GAMS is accessible through a web front-end and allows users to search for mathematical software that can solve various problems. GAMS is not a web service but illustrates the needs of the community in terms of distributed computing support for contacting software that can solve mathematical problems. For any given problem, GAMS proposes a way to find the solution (e.g. "run Maple and enter the following commands"). GAMS can be viewed as a simple MSM.

The MONET project creates a framework for describing problems solved by given services. This framework is flexible enough to accept classifications such as GAMS, but also provides a way to precisely specify what problem is solved by a given service (with information such as constraints and relations to other problems). The Mathematical Problem Description Library (MPDL) is a repository for mathematical problems. Each problem is a document including well-defined delimiters (typically XML tags) and supporting various formalisms for describing manipulated objects (for example, OpenMath). The Mathematical Service

Description Language (MSDL) [10] provides support for describing mathematical services available in a given environment (usually on the Web). This language is used to create a document to link low-level considerations described by a WSDL [11] document to semantic information expressed by a GAMS or an MPDL entry. Using such classifications, it is possible to provide a service matcher with sufficient information for selecting services. In this situation, there is not only a question of discovering a service that can solve the problem, but also selecting the most appropriate service. Semantic data helps with this selection. When the client submits a problem instance to the service matcher, this problem is expressed using references to entries in databases and repositories both for the problem itself and its inputs. The service matcher can then query its base of registered services and list entries that best fit the request.

We now present an example of a problem-oriented description of a mathematical web service using the MONET language. A first step is to define the problem itself using one type of classification. We choose to provide an entry of a problem description library as defined in the MONET project. The problem we describe here is `factorInteger`, which factors an integer and returns the list of its primes:

```
<definitions
    xmlns="http://monet.nag.co.uk/monet/ns"
    xmlns:om="http://www.openmath.org/OpenMath">

<problem name=factorInteger>
  <documentation>
   factors an integer and returns the list of its primes.
  </documentation>
  <header/>
  <body>

    <input name=n> Integer Type </input>
    <output name=p> List of Primes </output>

  </body>
</problem>

</definitions>
```

The documentation provides informal semantics of the problem. The type of input and output values is expressed using the OpenMath formalism (not shown here and replaced by type names like *Integer Type*). At this point, this problem description does not have any relation with a web service. It is merely the description of a problem, that could be replaced by an RDF description or a reference to a GAMS entry.

The next step is the MSDL document. At this point, it is important to note that this language is very flexible and allows many ways to describe a service. We provide here a limited but practical view of what appears in a document written using such a language.

```
<definitions
    xmlns="http://monet.nag.co.uk/monet/ns"
    xmlns:om="http://www.openmath.org/OpenMath">

<service name="factorInt">
  <classifications>
    <problem-reference>
      http://monet.nag.co.uk/pdl/factorization.pdl
    </problem-reference>
  </classifications>

  <implementation>
    <software>
     http://www.medicis.polytechnique.fr/Software/PARI
    </software>
    <algorithm>
      http://monet.nag.co.uk/algorithms/intfact.adl#Shanks
    </algorithm>
  </implementation>

  <service-interface-description
   sid-ref="http://www.medicis.polytechnique.fr/PARI/pari.wsdl"/>

  <service-binding>
    <map operation="factorint" problem-ref="factorInteger" />
    <message-construction
       io-ref="n"
       message-name="factorintRequest"
       message-part="in0" />
    <message-construction
       io-ref="p"
       message-name="factorintResponse"
       message-part="factorintReturn" />
  </service-binding>
</service>
</definitions>
```

In this document, the service is described by relating a semantic description of the problem to an implementation (the PARI [12] software) solving (instances of) this problem. The implementation is usually represented by a WSDL document (`pari.wsdl` in the example), while the semantic description is a reference to the PDL entry. The mapping between both elements is done in the `service-binding` section.

4 Mathematical service brokering and composition planning

The Mathematical Service Matcher can be viewed as a regular service that implements a special functionality in the context of web services. It is a service that gathers information on other services and makes deductions about them. The usefulness of this service could be further extended by integrating modules to control services' life-cycles and to increase the reasoning power through communication with specialized services. The resulting service is called a broker (as defined by the MathBroker [13] or the MONET projects) and may contain several components: a service matcher, but also an execution manager as well as a planning manager. These last two elements are described in this section.

4.1 Planning and composition of services

Finding an appropriate service for a given query is the job of the Service Matcher that would lookup the Registry and return information on a service solving the submitted problem. Unfortunately, a match is never guaranteed to be found, this may happen for two main reasons:

- no service solving this problem has been registered yet.
- there exists no service solving this problem.

The issue is similar in the context of interactive mathematical systems and with libraries. There exists a number of routines available, but users almost never make just one call of such routines. They rather use a language provided by the systems to combine the calls of such routines in more complete algorithms suited to their needs. Mathematical web services only offer interfaces to invoke one routine or another. We propose to get back the flexibility and usability of known interactive systems by using the planning environment. A planning language would allow end-users to build programs combining routine calls. Such programs would be analyzed for optimization by the Planning Manager and executed by the Execution Manager.

Plans can be obtained from two sources: human and program. We notice that, even in a specific area of mathematics, being able to automatically analyze a problem and produce a plan "on-the-fly" is very difficult. Obviously, we can not expect the planning manager to solve any general problem. Most of the time, it cannot even decide by simply looking at a problem what domain is involved in a given query. This will depend on the classification chosen by the client to describe this problem. We can expect, however, that, with sufficient information, a program may be able to make deductions and produce a plan to combine solutions to sub-problems in order to produce the expected result for the submitted problem. We foresee that a more common scenario would be to use a human-made plans. People manually produce plans using a "mathematical plan language" and submit them to the Planning Manager. Also, note that automatically-generated plans may be submitted to the clients for approval.

Human-made plans are most likely to be widespread because they will cover any type of operation and may contain "tricks" that can not be inferred by a program. For convenience and reusability, human-made or automatically-generated plans can be gathered into repositories that the planning manager can query.

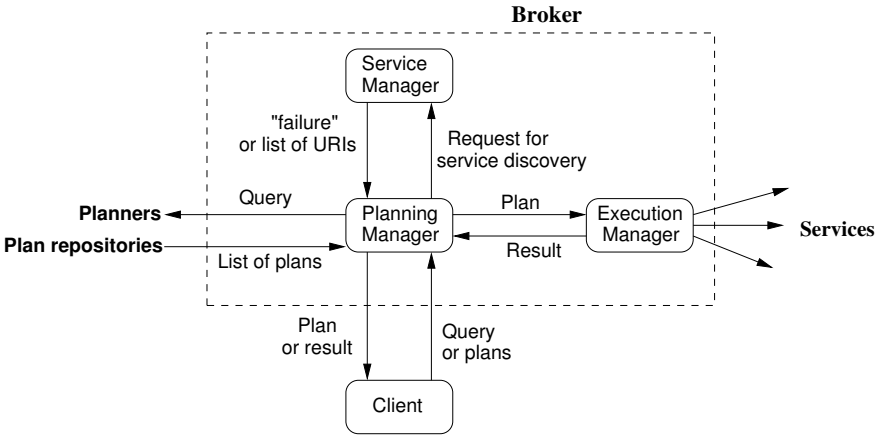


Fig. 3. MONET Architecture in the context of Planning

Figure 3 illustrates the architecture of the planning activity as organized in the MONET project. Clients send queries to the broker that uses a service matcher to locate appropriate services. If this does not succeed, the planning manager contacts planners to obtain mathematical plans to solve the problem and then creates plans of execution that match the constraints of the query. As a result of a successful planning activity, one or several execution plans are sent back to the client and, probably, proposed to the end user. The end user may then approve one of the plans or even carry out the plan himself/herself.

4.2 Examples of composition

Sequence of services invocations A specific type of simple analysis is to try and find a reasonable sequence of services invocations. These invocations are connected together through inputs and outputs. The output of a service would be used as input for the next one in the sequence. A typical example of such plan creation is a translation of mathematical object format: the client wants to convert an object from the OpenMath format into the Maple format. Unfortunately, the service matcher can only find two translation services: "OpenMath to MathML" and "MathML to Maple". In this case, deeper analysis would advise the client an execution plan that first invokes the OpenMath to MathML service on the original parameter and then the MathML to Maple service on the intermediate result.

Finding the real poles of a fraction We provide here an example of planning to solve a mathematical problem that would not typically be available as one operation exported by a package:

problem: "Find the real poles of"
input: $F(X) = \frac{X^2-X-2}{X^4+X^3-X^2+X-2}$

No package usually provides such operation, so the broker would not find any service returning the real poles of a fraction. However, it is likely that a plan exists to solve this problem. Here is an example of such a plan:

problem: "Find the real poles of a rational fraction $F(X) = \frac{N(X)}{D(X)}$"
steps:
1. extract $D(X)$ the denominator of $F(X)$
2. compute $S = \{X_1, X_2, ..., X_n\}$, the set of solutions of the equation $D(X) = 0$
3. extract the subset $R = \{X_{\alpha_1}, X_{\alpha_2}, ..., X_{\alpha_k}\}$ of the real values of S

Plans can thus be considered as an orchestration of various mathematical routines to create "on-the-fly" an otherwise unavailable service. In our example, an instantiation of the steps would be:

1. extract the denominator: $D(X) = X^4 + X^3 - X^2 + X - 2$
2. compute the solutions of $D(X) = 0$: $S = \{i, -i, 1, -2\}$
3. extract the subset R of the real values of S

This would lead to the result: $R = \{1, -2\}$.

4.3 Composition languages

Plans in the MONET framework are algorithms solving a problem. Queries sent by the client are typically instances of a problem and algorithms variables have to be mapped to actual values. Rather than choosing a language such as BPEL [14] for creating composition, we believe a higher level view is required to express strategies of computation. In particular, it is important in the context of brokering to be able to abstract plans and allow users to express algorithms rather than WSCI [15] documents linked to WSDL resources for example. Of course, an implementation of this system may use a known language, but we would to make it clear that this model is independent from the language. Therefore, we classify plans as follows:

– *Abstract* plans can be considered as "templates" (see [16]). Such templates contain two types of variables or unknowns: values manipulated by the process and methods/services using those values. In the context of mathematical computation, an abstract plan is really a parameterized algorithm, with no notion of web services. However, one exception would be fine-grained "configuration" of the process. It is reasonable to assume that a plan could specify certain constraints for a given step (e.g. "should be solved by the fastest service available"). In this case, a notion of computation by services could be introduced.

– *Resolved* plans are documents that can be directly translated into process documents for execution. For example, a resolved plan could be readily transformed into a BPEL document for execution into a BPEL engine. A resolved plan does not have any unknown variables (except for variables supposed to hold intermediate results) and each computing step is associated with a single service to invoke. Resolved plans are typically not stored, but are rather used for execution.
– *Unresolved* plans correspond to possible intermediate states between *abstract* and *resolved*. A valid unresolved plan contains bindings for certain parameters (variables or operations) but not all of them. Such a state may be useful for client to store plans in a repository with certain values already provided (in this case, we could compare such a technique to currifying techniques in functional programming). Also, certain steps of a plan may require the invocation of a specific well-known service, which would avoid invoking the service matcher for this step.

The actual planning language chosen for creating plans is not fixed. It would be perfectly acceptable to use the language from the Maple system for example. As long as the various entities of the system agree on using the same language, the architecture described by the MONET project is valid.

4.4 Executing services

Another task handled by the broker in the MONET framework is the execution of plans. The output of the Planning Manager is a list of possible plans. Each plan is annotated at each step by a **list** of available services to carry out that step. Execution plans can be handled by a client or by an "Execution Manager" when the client chooses to delegate this work. The Execution Manager is part of the broker and is responsible for executing plans. At an implementation level, there are two possibilities: either the Execution Manager directly executes a "resolved plan" handling concerns such as transactions, security, and so on, or or it relies on a third-party composition language such as BPEL into which the resolved plan is transformed. In both cases, for each step of the resolved plan, the Execution Manager may be responsible for selecting the service to invoke among the listed services shown in the annotation of each step of the plan. However, this task is typically done by the client. The list of services associated with each step of the plan would thus be used either by the client to select a preferred service based on criteria such as cost or location, or by the execution manager to choose a service hosted on a system with acceptable load for example.

We remark that, because the composition language is not fixed, BPEL could be replaced by the Maple language for example. This would require the Maple environment to support web services, but it is likely to be the case in the near future. Obviously, even abstract plans can be expressed using Maple, however we would like to discuss here the advantage of using a Maple system as an orchestration engine. Let us look again at the "Real poles" example in Section 4.2. There are three steps: extract denominator, solve equation, extract real values.

Two of these steps (extraction of denominator and real values) are very simple and it would be a waste of resources to invoke operations to perform these steps (if a service even decides to export such operations). This is a real problem, because, without those operations, the plan can not be carried out. In this case, a local computation engine such as Maple could help and perform these steps locally. The operation "solve equation" can then be done by any given service.

Local computation for trivial operations appears as a very useful feature of mathematical web services. Whenever an operation can be done locally (what these operations are can be decided at implementation, deployment or even runtime), a local computation system can be invoked by the execution engine. The advantage is also to reduce the number of lookups done by the Service Matcher (in our example, only a "solve equation" operation has to be searched for). Using Maple makes local computation very simple to setup, although we believe it should be possible to link computation engines to the execution manager or even a BPEL engine through a system of plug-ins.

5 Commercial exploitation

A natural exploitation plan for a company providing several mathematical packages or libraries would be to offer various types of access to a number of packages. We can easily imagine ranking mathematical routines from "free" to "expensive" depending on their quality, efficiency, use of top-of-the-line machine support, and so on. Certain brokers could go beyond semantic matching and planning by providing services similar to the ones provided by real-life insurance brokers: a broker could negotiate access to several services at a low price due to the number of clients accessing such services. Two types of businesses appear: brokering and servicing. Both can use the same framework (e.g. MONET), but implementors are free to offer whatever features they wish.

In the context of commercial exploitation, protocols are required to proceed with the payment for accessing a service. Payment could be done in a pay-per-use manner or with a sort of membership. It is rare that a user needs one day to solve, say, an ordinary differential equation without requiring further similar computation regularly. That is why we believe that the "membership" model will be the favorite one when using mathematical services. Note that the use of a broker and membership business model are entirely compatible. It is reasonable to assume that users having a membership with a company will want to use services of this company. However, such a company might also have a broker that provides access to many services. We believe that membership would cover the use of at least a group of services rather than a single specific one. In this case, a query sent with a membership code would handled by the broker and answers can be provided. Of course, this raises the question of security, which could be ensured via tools such as WS-security [17]. We remark, though, that as for orchestration of web services, security has not yet be standardized and it is likely that any solution could be chosen at this point.

As far as commercial exploitation is concerned, web services provide a new and more flexible way for institutions to market products. Let us take the case of a research center or a university. Many products of these environments are far beyond simple "alpha" version, and could very well be sold to the public. Sometimes this is not done simply because the cost and difficulty to market and distribute a piece of software is prohibitive. In the case of web services, and in the world of mathematics in particular, offering computation routines on a commercial basis would be possible. We can imagine that a research group provides access to its broker and all services for a fee covering the cost of maintaining quality access to such services. This allows individual services to become much more visible than they could have been as "stand-alone" applications. This situation is obviously not specific to mathematics, but is clearly one important aspect of dissemination of mathematical knowledge in today's industry. For example, analysis engine are difficult to construct and a possible business model would be to provide such an engine in the form of a MONET planner that can be contacted – for a fee – by the planning manager to obtain a plan of execution of quality.

6 From software to services

Because mathematics are so fundamental, numerous packages have been developed for both numerical and symbolic computation. Although created in an academic context, most of them are fully documented and regularly updated and some became well-known, commercial products, like Maple and Mathematica for symbolic computation, and the NAG libraries for numerical computation. On hot topics, one can find many different packages solving the same problem. For example, Gb [18], Singular [19], and Macaulay [20] offer very efficient routines to compute Groebner bases.

In the context of the MONET project, we are experimenting with a few symbolic mathematical packages, including Bernina, an interactive interface to the `Sum^it` [21] library. This library provides some efficient computations revolving around differential operators in $Q[x, d/dx]$ or $Q(x)[d/dx]$. A detailed presentation of this experiment can be found at [22].

Certain functions of Bernina would better benefit to the community if exposed as web services. Such functions – efficiently implementing non-trivial algorithms related to differential operators – solve very specific problems. Simple access to them (i.e. in a standard way and without installing the software) will increase the visibility and reachability of Bernina.

As for many existing applications, one obstacle for providing web service access to Bernina's functions is that they are implemented using Aldor [23] that does not provide web services functionalities like Java or a .NET language do. Consequently, they should be considered as closed source (i.e. a black box) and adding a web service capability to Bernina is difficult. Also, the formats (Maple or Lisp) of the manipulated objects are known but not standard. That can make using these functions more complicated within non-Maple-compliant software.

In our current experiments, our solution is to use MONET languages to expose Bernina's functions as mathematical web services. OpenMath has been selected to represent objects, because this is a standard language for this purpose. Finally, there is no need to modify the code of Bernina (even re-compilation is not necessary), because we are wrapping its functionalities into a Java program. The Java front-end allows us to "import" Bernina into the world of web services. In the OpenMath terminology, such a wrapper is called a *phrasebook*.

7 Conclusion

Most mathematical packages are mainly designed for interactive use. Linking these programs into a separate environment, or calling them from another package is difficult or sometimes impossible. Mathematical web services would solve this problem by potentially unifying all kinds of mathematical software, tools and services in a common framework where all components would be able to communicate with one another in an appropriate way. The main problem remains to create suitable taxonomies to describe unambiguously the various mathematical problems a mathematical software can solve. Another important issue is to provide tools to help users to complex problems using a combination of services.

Service-oriented computing applied to mathematics will have positive benefits on anybody using mathematical computation. This includes much of science and engineering, and indeed many other areas such as finance or health. Instead of developing self-contained systems needing large amount of infrastructure like user interface, help system or graphical output facilities, mathematical software manufacturers could concentrate on and produce small, highly-specialized applications which could be accessed through a variety of systems ranging from large problem solving environment to spreadsheet program. Service-oriented computing could allow the many small, specialized packages developed by the academic research community to be brought to market in a profitable way.

Acknowledgment

This work is partially supported by the European MONET project (IST-2001-34145).

References

1. The MONET Consortium, *MONET Home page: Mathematics on the Net*, `http://monet.nag.co.uk`
2. Maplesoft, `http://www.maplesoft.com`
3. Wolfram Research, `http://www.wolfram.com/products/mathematica`
4. Manuel Bronstein, *The BERNINA User Guide*, `http://www-sop.inria.fr/cafe/Manuel.Bronstein/sumit/berninadoc`
5. National Institute of Standards and Technology, *GAMS: Guide to Available Mathematical Software*, `http://gams.nist.gov`

6. OASIS, *Universal Description, Discovery and Integration of Web Services*, `http://www.uddi.org/`.
7. Mike Dewar, David Carlisle, Olga Caprotti, *Description Schemes For Mathematical Web Services*, Electronic Workshops in Computing, Oxford, 2002.
8. John Abbott , Angel Diaz , Robert S. Sutor, *A report on OpenMath: a protocol for the exchange of mathematical information*, In ACM SIGSAM Bulletin, volume 30, number 1, pages 21-24, 1996.
9. W3C, *W3C Math Home*, `http://www.w3.org/Math`
10. Stephen Buswell, Olga Caprotti, Mike Dewar, *Mathematical Service Description Language: Final Version*, The MONET Consortium, Deliverable D14. Available from `http://monet.nag.co.uk`
11. W3C, *Web Services Description Language (WSDL) 1.1*, `http://www.w3.org/TR/wsdl.html`.
12. Henri Cohen & al, *PARI/GP*, `http://www.math.u-psud.fr/~belabas/pari`.
13. Olga Caprotti, Wolfgang Schreiner, *MathBroker overview*, Project Report, RISC-Linz, Johannes Kepler University, Linz, Austria, November 2002.
14. BEA Systems, IBM, Microsoft, SAP AG, Siebel Systems, *Business Process Execution Language for Web Services version 1.1*, Available from `http://www-106.ibm.com/developerworks/library/ws-bpel`
15. BEA Systems, Intalio, SAP AG, Sun Microsystems, *Web Service Choreography Interface (WSCI) 1.0 Specification*, Available from `http://wwws.sun.com/software/xml/developers/wsci`
16. Biplav Srivastava, Jana Koehfler, *Web Service Composition – Current Solutions and Open Problems*, Workshop on Planning for Web Services, Trento, 2003, available at `http://www.isi.edu/info-agents/workshops/icaps2003-p4ws/program.html`).
17. IBM, Microsoft, Verisign, *Specification: Web Services Security (WS-Security) Version 1.0*, Available from `http://www-106.ibm.com/developerworks/library/ws-secure`
18. Jean-Charles Faugère, *Gb*, `http://www-calfor.lip6.fr/~jcf/Software/Gb`
19. G.-M. Greuel, G. Pfister, and H. Schönemann, SINGULAR 2.0. A Computer Algebra System for Polynomial Computations. Centre for Computer Algebra, University of Kaiserslautern (2001). `http://www.singular.uni-kl.de`.
20. David Eisenbud, Daniel R. Grayson, Michael E. Stillman, Bernd Sturmfels, *Computations in algebraic geometry with Macaulay 2*, Springer-Verlag, September, 2001, n.8 in "Algorithms and Computations in Mathematics", ISBN 3-540-42230-7.
21. Manuel Bronstein, *SUM-IT: A strongly-typed embeddable computer algebra library*, in Proceedings of DISCO'96, Springer LNCS 1128, 22-33.
22. Yannis Chicha, Marc Gaëtano, *Putting Bernina on the Web*, Mathematics on the Semantic Web Workshop, Eindhoven, 2003.
23. Aldor.org, `http://www.aldor.org`, 2001-2003.

An ontology-based method for classifying and searching *e*-Services ⋆

D. Bianchini, V. De Antonellis and M. Melchiori

Università di Brescia
Dip. Elettronica per l'Automazione
Via Branze, 38
25123 Brescia - Italy
bianchin|deantone|melchior@ing.unibs.it

Abstract. While the Service Oriented Architecture paradigm is becoming mature and *e*-Services are being made available on the Web, there is the need for organizing them according to semantic aspects for discovery purposes. In this paper, we propose an ontology-based method to classify *e*-Services into three different layers according to semantic relationships that can be semi-automatically derived from the *e*-Service descriptions. The resulting ontology architecture can be exploited to allow the search and discovery of services on the basis of the defined semantic relationships.

1 Introduction

The Web is rapidly evolving its functionality from document to service oriented. *E*-Services offered on the Web are independently published by different providers and, according to the Service Oriented Architecture (SOA [8]), users can request and access a wide variety of *e*-Services with desired capabilities. *E*-Services satisfying user preferences and constraints are selected, possibly by means of brokers, looking for them in a Service Registry. To facilitate service discovery two main research issues have to be addressed: (i) services organization and classification according to semantic properties at different levels of abstraction; (ii) service semantic search on the basis of their essential features.

In this framework, the availability of an *e*-Service conceptual model and the construction of *e*-Service ontologies are particularly relevant.

DAML-S [2] has been proposed in the Semantic Web community for semantic description of *e*-Services whose content is expressed in terms of domain concepts contained in a domain ontology. DAML-S has been adopted in several proposals in the literature [12–14]. A series of DAML-S weak points due to the imprecise underlying conceptual model are discussed in [15].

In the Italian MAIS (Multichannel Adaptive Information Systems) project [10] we have defined a comprehensive modeling approach for supporting dynamic and

⋆ This work has been partially supported by the Italian VISPO (Virtual district Internet-based Service PlatfOrm [16]) and MAIS (Multichannel Adaptive Information Systems [10]) projects.

flexible composition of *e*-Services in a multichannel environment. The proposed conceptual model [1, 5] covers: (i) *functional aspects*, in terms of operations, I/O parameters, pre- and post-conditions; (ii) *non-functional aspects*, in terms of quality dimensions; (iii) *specific contextual aspects*, in terms of user preferences, location, devices and channels.

In this paper, for service classification we focus on functional and behavioral aspects and abstract them into I/O descriptors and pre-/post-conditions [3]. Our aim is to provide a semi-automated method to build *e*-Service ontologies in a domain, where similarity, equivalence and generalization relationships among services are established. Given the ontology, service semantic search can be performed by exploiting the defined semantic relationships.

This paper is organized as follows. Section 2 proposes a conceptual model for *e*-Services. Section 3 introduces a reference example to show an application of our approach. In Sections 4 and 5 our approach to organize and classify *e*-Services in a Service Ontology is presented. Section 6 explains how service semantic search is performed by exploiting the Service Ontology. Finally, conclusions and future work are discussed.

2 *E*-Service Model

For classification and discovery purposes, the description of the *e*-Service capabilities needs to express functional aspects ("what the *e*-Service does", in terms of operations, input and output properties, pre-/post-conditions) and non-functional aspects (for instance, features describing the quality of service). The *e*-Service description has to be understandable by humans as well as by machines, providing such descriptions both at syntactic and at semantic level. A part of the MAIS *e*-Service model, here limited to the functional description, is shown in Figure 1.

An *e*-Service is described by means of a `name` that identifies it, a short textual `description` that helps human reader to understand what the service does and a service `category` (e.g., *Financial and Insurance service* or *Travel service*), according to the classification proposed by UNSPSC [7]. An *e*-Service is also characterized by a `FunctionalDescription`, composed by:

- a set of `Operations`, with a `name` and a short textual `description` to help the human reader to understand "what the operation does";
- each operation requires one or more `Inputs` and gives back one or more `Outputs`; input and output parameters are described by means of a `name` and a range of admitted `values`;
- each operation is associated with a set of `Pre-conditions`, which must be verified before the execution of the operation, and a set of `Post-conditions`, which must be satisfied after the execution; each `Pre-` and `Post-condition` is characterized by a boolean `expression` representing the statement that must be verified; pre- and post-conditions can be assigned also to the whole *e*-Service;

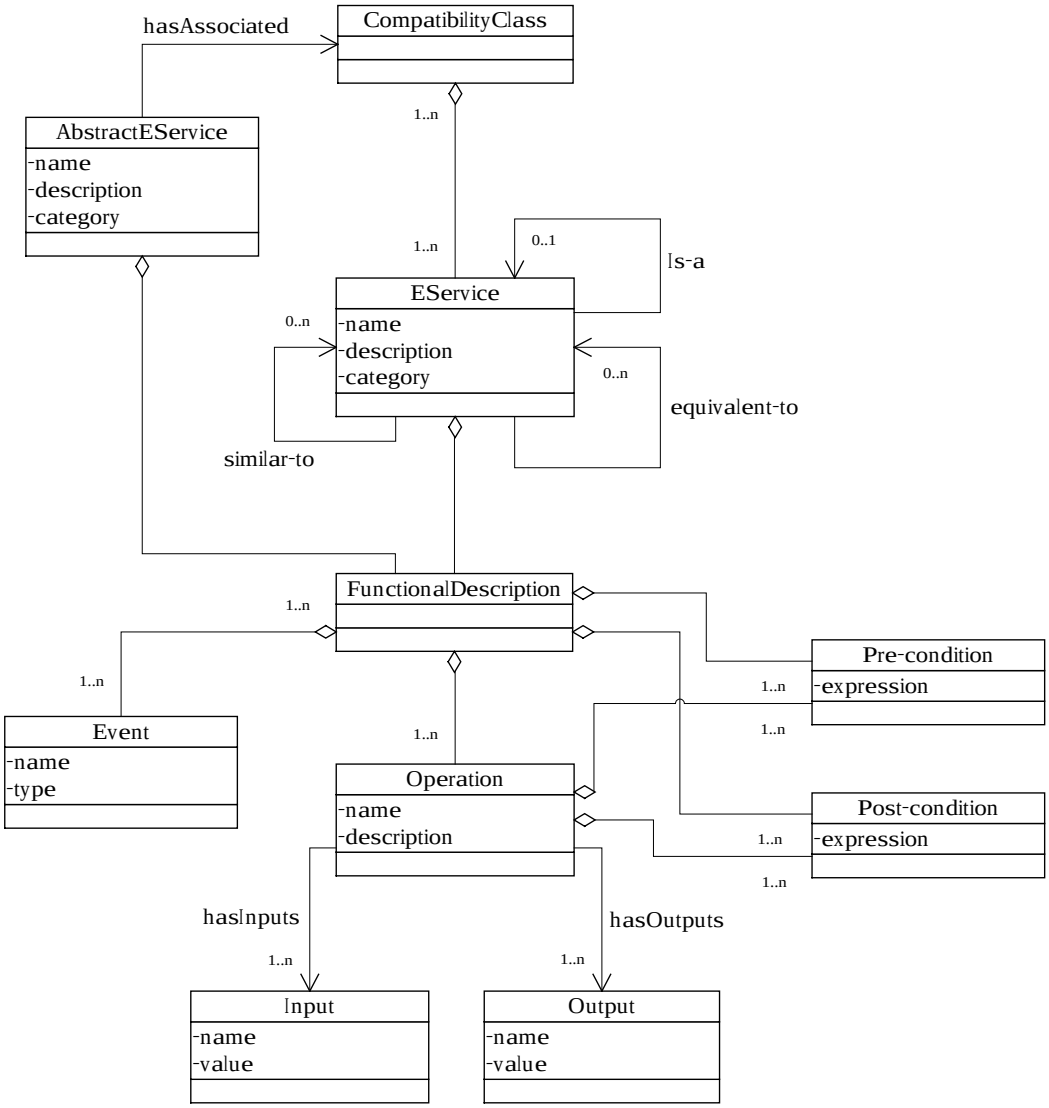


Fig. 1. *E*-Service conceptual model

– external `Events` are used to model actions that are asynchronous with respect to the normal flow of the *e*-Service; each event has a `name` that identifies it and a `type` such as *temporal* or *data event* (an example of *temporal event* is a timeout that occurs during the execution of an operation).

Three types of semantic relationships between *e*-Services (*is-a, similar-to, equivalent-to*) are established through evaluation of similarity coefficients as explained in Sections 4 and 5. Furthermore, *e*-Services are grouped into `CompatibilityClasses` for substitutability purposes. In fact, a compatibility class is associated with an `AbstractEService`, that is used to specify the required features of a service in a process-based composition [3].

3 Reference example

In this section, a reference example is introduced to illustrate the main aspects of our approach to the classification and search of services. We consider four *e*-Services in the domain of the weather that allow the users to ask for weather forecasts and other related services. According to the functional model introduced in the previous section, for each service we describe the operations that the service provides, its input and output entities and its pre- and post-conditions, as it is shown in Table 1.

In particular, the first service (called *Simple Weather Forecast service*) requires that a valid account is specified through the userAuthentication operation and provides the description of the weather forecast and the temperature range for a given city in a given country (through the execution of the weatherForecast operation). The service pre-condition, described by means of a propositional logical formalism, requires that the account given as input is valid and that the given city belongs to the specified country. The post-condition describes the effects of service execution: that is, the output parameters describe the weather forecast, the high temperature, the low temperature in the given city for the following day.

The second service (*Weather Forecast service*), besides the weather forecasts and the high and the low temperatures, provides also weather maps. The pre-condition requires that the input specified city is in the given country. The post-condition describes the output: forecastDescription, minTemp and maxTemp are respectively the wheater forecast, the high temperature and the low temperature for the following day for the specified city. The map parameter is the weather map for the specified input country.

The third service (*Worldwide Weather service*) has a single operation 48Hours-Forecast that takes as input a city name (or, as an alternative, a ZIP code) and a country name, and returns a weather forecast, valid for the next 48 hours, and the expected low and high temperatures. The post-condition describe the results as in the previous services, while the pre-condition requires that either the city or the ZIP code are related to the country.

Finally, the *Weather Photo&Map Archive service* allows users to access to a repository of satellite weather photos and weather maps (for every day since

01/01/1998 to the current date). In particular, the getSatellitePhoto operation takes as input a region and a date and returns the photo of the region in the given date. Analogously, the getWeatherMap operation returns the weather map of the specified region in the required date. Pre-condition states that input parameters are to be valid, while post-condition describes that the output maps or photos are correctly related to the input region and date.

Simple Weather Forecast service		
Operation	**I/O entities**	**Pre- and Post Conditions**
userAuthentication	I=user,password O=answer	PreC≡ belongs(city,country) and validAccount(user, password); PostC≡ forecastDescription = tomorrowWeather(city, country) and highTemperature = tomorrowHighTemperature(city, country) and lowTemperature = tomorrowLowTemperature(city, country);
weatherForecast	I=city,country O=forecastDescription, highTemperature, lowTemperature	

Weather Forecast service		
Operation	**I/O entities**	**Pre- and Post Conditions**
weatherForecast	I=city,country O=forecastDescription, map, minTemp, maxTemp	PreC≡ belongs(city,country); PostC≡ forecastDescription = tomorrowWeather(city, country) and map = subject(country) and isWeatherMap(map) and minTemp = tomorrowHighTemperature(city, country) and maxTemp = tomorrowLowTemperature(city, country);

Worldwide Weather service		
Operation	**I/O entities**	**Pre- and Post Conditions**
48HoursForecast	I=city,ZIP,country O=description, high, low	PreC≡ (belongs(city,country) or belongs(ZIP,country)); PostC≡ ((description= 48HoursForecast(city, country)) and (high = 48HoursHighTemperature(city, country)) and (low = 48HoursLowTemperature(city, country))) or ((description=48HoursForecast(ZIP, country)) and (high = 48HoursHighTemperature(ZIP, country)) and (low = 48HoursLowTemperature(ZIP, country)));

Weather Photo&Map Archive service		
Operation	**I/O entities**	**Pre- and Post Conditions**
getSatellitePhoto	I=region, date O=photo	PreC≡ validRegion(region) and ValidDate(date) and (01/01/1998≤ date ≤ currentDate()); PostC≡ ((map = subjectRegion(region)) and (date = MapDate(map)) or ((photo = subjectRegion(region)) and (date = PhotoDate(photo)));
getWeatherMap	I=region, date O=map	

Table 1. Functional descriptions for the example services

4 *E*-Service semantic similarities

In this section, we present an approach to analyze *e*-Services and establish semantic relationships among them for the service ontology construction. To this purpose, we define criteria and techniques for *e*-Service semantic analysis. This analysis is performed on properly defined Descriptors providing information on essential *e*-Service features.

Specification of the e-*Service interface with I/O Descriptors.* A service I/O descriptor provides a summary, structured representation of the features of an *e*-Service that are relevant for similarity assessment.
A descriptor is formally described as a service name and a set of triplets:

$$\langle \text{operation } (OP), \text{ input entities } (IN), \text{ output entities } (OUT) \rangle$$

Example 1. The descriptor for the `Simple Weather Forecast service` obtained from its functional description is the following:

```
SERVICE     Simple Weather Forecast
OPERATION   userAuthentication
    INPUT   user
    INPUT   password
    OUTPUT  answer
OPERATION   weatherForecast
    INPUT   city
    INPUT   country
    OUTPUT  forecastDescription
    OUTPUT  highTemperature
    OUTPUT  lowTemperature
```

Specification of the e-*Service behavior with Pre- and Post-condition Descriptors.* The behavior of an *e*-Service is described by pre-condition and post-condition descriptors that give a characterization of the *e*-Service semantics and in particular provide complementary information with respect to the I/O descriptors. In general, pre- and post-conditions can be associated to the whole service but also to the single operations of the service. For classification and searching purposes, in this paper we consider pre- and post-conditions associated to the whole service. For composition and substitutability purposes, pre- and post-conditions at single operation level are considered in [3].

4.1 Interface similarity analysis

Given the descriptors of two services, we establish their similarity according to the ratio of (i) similar input/output entities and (ii) similar operations. For this purpose, we compare *e*-Services on the basis of the analysis of their descriptors and therefore of their interfaces. *E*-Services are matched and clustered with respect to similarity coefficients computed by the ARTEMIS tool environment [3, 4] and presented in the following. A more extended discussion on the feasibility of the similarity analysis and relevant experimentation have been presented in [6].

– *Entity-based similarity coefficient.* The Entity-based similarity coefficient of two *e*-Services S_i and S_j, denoted by $ESim(S_i, S_j)$, is evaluated by comparing the input/output information entities in their corresponding descriptors.

In particular, names of input and output entities are compared to evaluate their degree of affinity $A()$ (with $A() \in [0,1]$). The affinity A() between names is computed exploiting a thesaurus of weighted terminological relationships (e.g., synonymy, hyperonymy). To cover the terminology used in the descriptors two different alternatives are possible in ARTEMIS: to use a *pre-existing, domain independent basic ontology*, such as WordNet, or to use an *hybrid ontology* that is a thesaurus containing both terminological relationships extracted from WordNet and terminological relationships supplied by the domain expert.
Two names n and n' of entities have affinity if there exists at least one path of terminological relationships in the thesaurus between n and n' and the strength of path is greater or equal to a given threshold.
The higher the number of pairs of entities, one from the first service and one from the second, with affinity, the higher the value of $ESim$ for the considered e-Services.

– *Functionality-based similarity coefficient.* The Functionality-based similarity coefficient of two e-Services S_i and S_j, denoted by $FSim(S_i, S_j)$, is evaluated by comparing the operations in their corresponding descriptors. Also in this case, the comparison is based on the affinity $A()$ function.
Two operations are similar if their names, their input information entities and output information entities have affinity in the thesaurus. The similarity value of two operations is obtained by summing up the affinity values of their corresponding elements in the descriptors.
The value of $FSim$ coefficient is such that the higher the number of pairs of operations, one from the first service and one from the second, with similarity, the higher its value for the considered e-Services.

Finally, a global similarity coefficient $GSim$ for each pair of services S_i and S_j is evaluated by taking a weighed sum of $ESim(S_i, S_j)$ and $FSim(S_i, S_j)$, that is a measure of their level of overall similarity.

Example 2. Table 2 presents the results of the interface similarity analysis, evaluated, with the support of ARTEMIS, on the services of our example. In particular, the e-Services *Simple Weather Forecast service*, *Weather Forecast service* and *Worldwide Weather Service* show an high similarity because of the relevant ratio of input entities with affinity, output entities with affinity and the similarity of their operation (names and parameters). On the contrary, the e-Service *Weather Photo&Map Archive service* has low similarity with the other ones, since it has few pairs of Input/Output entities that have affinity with Input/Output entities of the other services and no similar operation.

4.2 Behavior similarity analysis

The behavior-based analysis complements the interface similarity analysis taking into account also the information on the behavior of the whole service in terms

e-Service1	*e*-Service2	GSIM
Simple Weather Forecast service	*Weather Forecast service*	0.7
Simple Weather Forecast service	*Worldwide Weather Service*	0.6
Simple Weather Forecast service	*Weather Photo&Map Archive service*	0.1
Weather Forecast service	*Worldwide Weather Service*	0.7
Weather Forecast service	*Weather Photo&Map Archive service*	0.2
Worldwide Weather Service	*Weather Photo&Map Archive service*	0.1

Table 2. Global similarity coefficients for the example services

of pre- and post-conditions. The analysis relies on the ideas of [17] for the specification and matching of software components, adapted to establish similarity relationships among *e*-Services on the basis of service behavior analysis.

A service S_i has *exact-behavior-similarity* with S_j (S_i *E-BSIM* S_j) if: (i) they are similar according to the interface similarity analysis; (ii) the pre-conditions of S_i are logically equivalent to the pre-conditions of S_j; (iii) the post-conditions of S_i are logically equivalent to the post-conditions of S_j. The rationale is that the services match if their descriptors are similar and the pre- and post-conditions are equivalent, that is, we could use S_i in place of S_j and viceversa. However, the *exact-behavior-similarity* is a strict requirement, so a weaker relation is introduced.

A service S_i has *partial-behavior-similarity* with S_j (S_i *P-BSIM* S_j) if: (i) they are similar according to the interface similarity analysis; (ii) the pre-conditions of S_i logically imply the pre-conditions of S_j; (iii) the post-conditions of S_j logically imply the post-conditions of S_i.

The rationale is that S_i partially matches S_j if their descriptors are similar and the execution of S_j can substitute the execution of S_i. In fact, if the pre-conditions of S_i are satisfied before its execution then also S_j pre-conditions are satified, and the execution of S_j has effects that make true the S_j post-conditions and therefore satisfy also the S_i post-conditions.

Example 3. In our reference example no *exact-behavior-similarities* are established since there is no couple of services with logically equivalent pre- and post-conditions. Instead, a partial match is present: *Simple Weather Forecast service* has *partial-behavior-similarity* with *Weather Forecast service* since the following conditions are valid: (i) *Weather Forecast* is similar to *Simple Weather Forecast* according to the interface similarity analysis; (ii) pre-condition of *Simple Weather Forecast* logically implies the pre-condition of the *Weather Forecast service*; (iii) post-condition of *Weather Forecast* logically implies the post-condition of the *Simple Weather Forecast service*. The intuitive meaning is that the *Weather Forecast* output includes the output of *Simple Weather Forecast* and the input of *Simple Weather Forecast* is suitable to run correctly the *Weather Forecast*; that is, *Weather Forecast* can replace *Simple Weather Forecast*.

5 Ontology-based *e*-Service semantic classification

According to similarity analysis, we are able to introduce semantic relationships among *e*-Services and classify *e*-Services on the basis of their descriptions.

The result of this classification is a domain service ontology that organizes *e*-Service specifications in different layers (see Figure 2).

Interface similarity layer. In the interface similarity layer, we find sets (clusters) of semantically related *e*-Services on the basis of measured similarity of their interfaces. *E*-Services are matched and clustered with respect to the global similarity coefficients that are computed by the ARTEMIS tool environment [4].

In ARTEMIS a hierarchical clustering algorithm [9] is used to determine clusters based on the strength of global similarity established among *e*-Services. In particular, similarity thresholds can be properly set and experimented in the ARTEMIS tool environment to provide different levels of compatibility under different perspectives. Between each pair of services S_i and S_j occurring in the same cluster, a S_i `similar-to` S_j relationship is added in this layer.

Behavior similarity layer. In the behavior similarity layer, we organize *e*-Services according to behavior-similarity relationships. Association links are maintained between *e*-Services in this layer and corresponding clusters in the interface similarity layer. The behavior similarity layer is constituted by *e*-Services and semantic relationships among them, according to the following rules:

- if (S_i *E-BSIM* S_j) holds, then a S_i `equivalent-to` S_j relationship is added in this layer;
- if (S_i *P-BSIM* S_j) holds and there is no *E-BSIM* between S_i and S_j, then a S_j `is-a` S_i relationship is added in this layer.

Category layer. In the category layer, service categories provide topic-based views of the *e*-Services considered in the ontology and are useful to guide the *e*-Service searching. We adopt the UNSPSC taxonomy as the hierarchy of service categories to classify the *e*-Services in our ontology. The UNSPSC taxonomy is structured as four hierarchical levels of categories. Association links are mainteined between a leaf category and the *e*-Services belonging to it in the underlying layers. In particular, if we consider a group of *e*-Services in the behavior similarity layer, that are all connected by a path of relationships (*is-a, equivalent-to*), we choose the more general *e*-Service (if one) or one of the more general *e*-Services and associate it to the corresponding category. If an *e*-Service is not represented in the behavior similarity layer, an association link connects its occurrence in the interface similarity layer to the corresponding category in the category layer.

Example 4. To illustrate an example of service classification into the domain service ontology, we refer to our reference example. The resulting portion of ontology, concerning to the four weather forecast services, is shown in Figure 2.

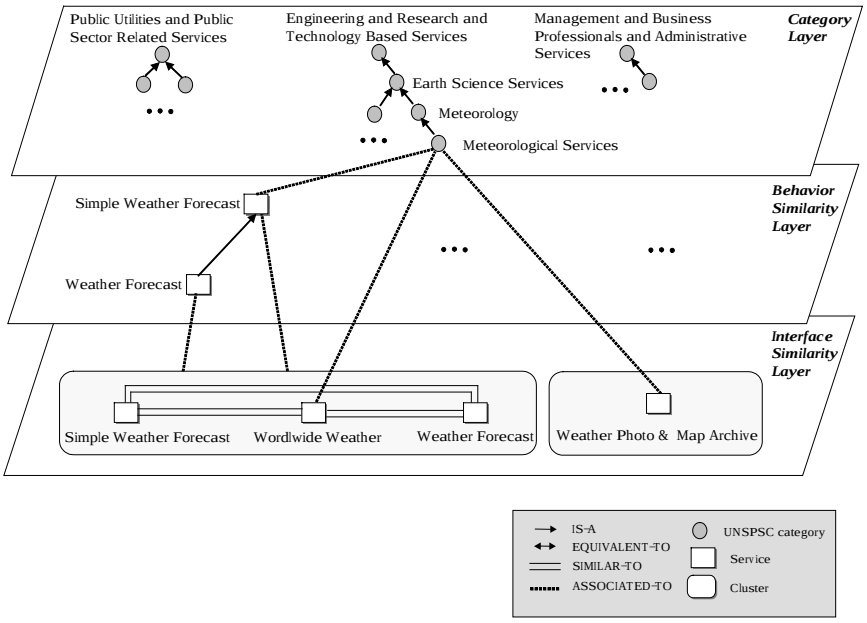


Fig. 2. Example of domain service ontology.

The clusters of services obtained by means of the clustering functionality of ARTEMIS (posing the threshold to select clusters equal to 0.6) and reported in the Interface Similarity Layer are { *Simple Weather Forecast, Weather Forecast, WorldWide Weather*} and {*Weather Photo & Map Archive* }. The composition of the first cluster indicates that *Simple Weather Forecast service*, *Weather Forecast service* and *Worldwide Weather service* are very similar. At the behavior similarity layer a *Weather Forecast* `is-a` *Simple Weather Forecast* relationship is added since *Weather Forecast service* has a partial-behavior-similarity with *Simple Weather Forecast service*. The Category Layer is illustrated by showing a portion of the UNSPSC taxonomy with the category to which the weather *e*-Services belong.

6 Ontology-based *e*-Service semantic search

The *e*-Service semantic search is performed by matchmaking a user service request against *e*-Services descriptions in the *e*-Service Ontology. In literature, the problem of searching services through matchmaking between service requirements and advertisement has been addressed in several approaches (see for example [11] and [14], where both functional and non-functional features are considered). In [13] requirements and suggestions about *e*-Service matchmaking process are provided. In this section, we present our approach based on the use

of the different semantic relationships in the Service Ontology. In the Service Ontology an *e*-Service is described by a *frame* as illustrated in Table 3.

Feature	Description
Name	The name of the *e*-Service
Category	The list of categories in which the *e*-Service is classified
Operations	A set of 5-uples ⟨`operation name, inputs, outputs, pre-conditions, post-conditions`⟩, one for each *e*-Service operation
Pre-conditions	The set of pre-conditions applied to the whole *e*-Service
Post-conditions	The set of post-conditions applied to the whole *e*-Service
Is-a	The names of *e*-Services with which the current one has an *is-a* relationship (that generalize the current one)
Equivalent-to	The names of *e*-Services with which the current one has an *equivalent-to* relationship
Similar-to	The names of *e*-Services with which the current one has a *similar-to* relationship
Web-link	The URL of the *e*-Service

Table 3. *E*-Service frame structure.

Each *e*-Service is characterized by means of a set of features that are exploited during the matchmaking process: the name of the *e*-Service, one or more categories in which the *e*-Service is classified (according to UNSPSC taxonomy), the description of its operations (with name, inputs, outputs, pre- and post-conditions), the sets of pre- and post-conditions associated to the whole *e*-Service and the name of other *e*-Services with which it has a *is-a*, *equivalent-to* or *similar-to* relationship.

Example 5. The *Simple Weather Forecast service* in Table 1 could be represented through the following frame structure:

Name:	Simple Weather Forecast
Categories:	{Meteorological services}
Operations:	{⟨weatherForecast, {city, country}, {forecastDescription, highTemperature, lowTemperature}, { }, { }⟩, ⟨userAuthentication, {user, password}, {answer}, { }, { }⟩ }
Pre-conditions:	{belongs(city,country) and validAccount(user, password)}
Post-conditions:	{forecastDescription = tomorrowWeather(city, country) and highTemperature = tomorrowHighTemperature(city, country) and lowTemperature = tomorrowLowTemperature(city, country)}
Is-a:	{}
Equivalent-to	{}
Similar-to	{WorldWide Weather, Weather Forecast}
Web-link	{http://www.nws.noaa.gov/}

Looking for a certain *e*-Service, the *e*-Service Ontology can be used with different modalities by exploiting the relationships between *e*-Services.

Category-driven search. The user can browse the service categories in the UNSPSC taxonomy and find *e*-Services classified in such categories.

Keyword-driven search. Users can specify one or more keywords to locate *e*-Services concerned with such topics. The submitted keywords are matched against the *e*-Service names, the names of their operations and their input and output entities with the help of ARTEMIS basic ontology of terminological relationships (*synonyms, hypernyms* and *hyponyms*). The user can formulate a query with one or more keywords to specify what *e*-Services he's looking for. The Domain Knowledge Ontology is exploited to find terms used to describe the *e*-Services and such terms are then matched with names of *e*-Services, operations, input and output parameters in the Service Ontology to find those the user is searching.

Similarity-driven search. One of the advantages in using an ontology for searching purposes is the possibility to go beyond the syntactic description of a service by considering also semantic aspects. Exploiting the semantic relationships (*is-a, equivalent-to, similar-to*) it is possible to find a set of semantically related *e*-Services from a functional point of view. Semantic relationships express different levels of interface and behavior similarity and provide users with the capability of selecting within this set the *e*-Service best fitting other non functional requirements. For example, among several functionally similar *e*-Services, an user can select that are provided on a desired channel (e.g., Web) with desired quality levels.

Example 6. Suppose that the user browses the UNSPSC taxonomy and navigates using a category-driven search

```
Engineering and Research and Technology Based services
    Earth Science services
        Meteorology
            Meteorological services
```

The `Meteorological services` category is chosen and the *e*-Services shown in Table 1 are presented to the user.

Example 7. Now suppose that the user specifies the following keywords

```
weather + forecast + temperature
```

In the basic Ontology the following terminological relationships are defined:

```
forecast BT weatherForecast
temperature BT highTemperature
temperature BT lowTemperature
forecast SYN forecastDescription
```

```
weather RT weatherForecast
```

where: `BT = BroaderThan, SYN = Synonymous, RT = RelatedTo`. A query

```
weather + forecast + temperature + weatherForecast +
highTemperature + lowTemperature + forecastDescription
```

is then formulated on the Service Ontology and it is matched with *e*-Service names, names of operations, input and output parameters. Suppose that the *e*-Services that better match this query are the *Simple Weather Forecast* and the *Weather Forecast service*. The user can start from these results and visit other *e*-Services through semantic relationships, as explained above.

7 Conclusion and future work

In this paper we have presented an ontology-based method for classifying and searching *e*-Services by means of semantic relationships of equivalence, similarity and generalization established in a semi-automated way. Exploiting these relationships, semantic search can be performed. Given an user request, it is possible to locate groups of *e*-Services satisfying it from the functional point of view. The choice of the best one can then be done on the basis of non-functional aspects. Specifically, one of the major issues to be considered in future work is related to the possibility of classifying *e*-Services according to selected non functional aspects, for example on the basis of categories of quality parameters and user profiles.

References

1. M. Adorni, S. Bandini, L. Baresi, D. Bianchini, V. De Antonellis, D. Micucci, B. Pernici, P. Plebani, C. Simone, G. Vizzarri, F. Tisato. Model Requirements: Architectural Model, Functional Model, Context Model, Metamodel. MAIS Technical Report R1.3.1, May 2003. `http://black.elet.polimi.it/mais/documenti/pdf/reportWP1_32.pdf`.
2. The DAML Services Coalition (A. Ankolenkar, M. Burstein, J. R. Hobbs, O. Lassila, D. L. Martin, D. McDermott, S. A. McIlraith, S. Narayanan, M. Paolucci, T. R. Payne, K. Sycara). DAML-S: Web Service Description for the Semantic Web. In *Proc. of the First Int. Semantic Web Conference, ISWC2002*, Sardinia, Italy, 2002.
3. V. De Antonellis, M. Melchiori, P. Plebani. An Approach to Web Service compatibility in cooperative processes. In *Proc. IEEE SAINT2003 of Int. Workshop on Services Oriented Computing: Models, Architectures and Application, SOC2003*, Orlando, Florida, USA, 2003.
4. The ARTEMIS Project Home Page. `http://www.ing.unibs.it/~deantone/interdata_tema3/Artemis/artemis.html`.
5. L. Baresi, D. Bianchini, V. De Antonellis, M.G. Fugini, B. Pernici, P. Plebani. Context-aware Composition of E-Services. In *Proc. of Fourth VLDB Workshop on*

Technologies for E-Services, TES2003, Humboldt-University zu Berlin (Germany), September 7-8, 2003.
6. S. Castano, V. De Antonellis, M. Melchiori. A Methodology and Tool Environment for Process Analysis and Reengineering. *Data and Knowledge Engineering*, 31(3):253–278, November 1999.
7. ECCMA. UNiversal Standard Products and Services Classification (UNSPSC). `http://www.eccma.org/unspsc/browse/`.
8. HP. Web Services concepts - A technical overview. `http://www.bluestone.com/downloads/pdf/web_services_tech_overview.pdf`.
9. A.K. Jain, R.C. Dubes. *Algorithms for Clustering Data*. Prentice-Hall, 1988.
10. The MAIS (Multichannel Adaptive Information Systems) Project Home Page. `http://black.elet.polimi.it/mais/index.php`.
11. E. Michael Maximilien, Munindar P. Singh. Agent-based Architecture for Autonomic Web Service Selection. In *Proc. of the Workshop on Web Services and Agent-based Engineering, AAMAS2003*, Melbourne, Australia, July 2003.
12. Sheila A. McIlraith, Tran Cao Son, Honglei Zeng. Mobilizing the Semantic Web with DAML-Enabled Web Services. In *Proc. of The Second International Workshop on the Semantic Web, SemWeb2001*, Hong Kong, China, May 2001.
13. Thomi Pilioura, Aphrodite Tsalgatidou, Alexandros Batsakis. Using WSDL/UDDI and DAML-S in Web Service Discovery. In *Proc. of WWW 2003 Workshop on E-Services and the Semantic Web, ESSW2003*, Budapest, Hungary, May 2003.
14. José M. Puyal, Eduardo Mena, Arantza Illarramendi. REMOTE: A Multiagent System to Select and Execute Remote Software Services in a Wireless Environment. In *Proc. of WWW 2003 Workshop on E-Services and the Semantic Web, ESSW2003*, Budapest, Hungary, May 2003.
15. Marta Sabou, Debbie Richards, Sander van Splunter. An experience report on using DAML-S. In *Proc. of WWW 2003 Workshop on E-Services and the Semantic Web, ESSW2003*, Budapest, Hungary, May 2003.
16. The VISPO Project Home Page. `http://cube-si.elet.polimi.it/vispo/index.htm`.
17. A. M. Zaremski, J.M. Wing. Specification matching of software components. In *Proc. the 3rd Int. ACM Symposium on Foundations of Software Engineering, SIGSOFT*, pages 6–17, Washington DC, USA, 1995.

A Service-Domain Based Approach to Computing Ambient Services

Thin Thin Naing, Seng Wai Loke, Shonali Krishnaswamy
School of Computer Science and Software Engineering
Monash University, Caulfield East, VIC 3145, Australia
{christine_naing@email.com, *swloke,shonali@csse.monash.edu.au}*

Abstract. Ambient services are related to the surrounding physical environment of the user, are locally useful and can be considered a form of location-based service. Ubiquitous short-range wireless networks provide a natural infrastructure for such services. Such services, by definition, have geographical boundaries or areas of relevance and usefulness. We call these areas service-domains. We discuss the stereotypical case of a user being in multiple service-domains at the same time, and how the set of suitable ambient services for the user can be computed using service-domains. We also describe our proof-of-concept prototype and its evaluation.

1 Introduction

An emerging view of applications is as services. Ideally, these services should be appropriate to the user's current environment. By *ambient services*, we have in view services that are related to the surrounding physical environment of the user, are locally useful (i.e., might not be relevant or useful beyond the boundaries of an area containing the user) and can be considered a form of location-based services. The emerging short-range wireless networking technologies [1,7,10] also tend to impose a natural geographical range restriction on services.

This paper discusses a concept to structure ambient services, and to compute, for a given user, what ambient services should be enabled for the user when in a given area, and in a given environment. This concept is the *service-domain*, which, associated with a given set of services, is the geographical area in which the set of services are available to users. We also describe a prototype system we have built to compute ambient services based on their service-domains.

We elaborate on service-domains in Section 2. A user might be within multiple service-domains at the same time (when the service-domains overlap), and so can utilize some combination of the services associated with each of the service-domains containing the user. Section 3 outlines our prototype system. Section 4 discusses related and future work, and concludes the paper.

2 Service-Domains

By definition, ambient services have service-domains. Ambient services are distinguished from other kinds of location-based services in that, for an ambient service, there is a geographical area (which we call the service's service-domain) outside of which the ambient service is judged not useful or relevant. This judgement is made by the developers of the ambient services and specified in tables which we show later.

The service-domains we consider in this paper will be of relatively fine granularity such as an office room, a lecture theatre or a building floor. A set of services is associated with each service-domain (e.g., specific library services while within a library, information services on specific museum exhibits of a museum hall, accessing particular printers when in some part of an office). When an individual is within a service-domain, the services associated with the service-domain can be invoked by the user. While an ambient service is being invoked and executed, if the user moves out of its service-domain, the system cannot guarantee successful completion of the service.

A user can be in several service-domains at the same time, and as the user enters one or more service-domains, his/her ambient services should be automatically discovered and enabled. Furthermore, as the user moves from one service-domain to another, these services should be changed or updated. An important issue to address in implementing such location based ambient services is how to decide what combination of services should be enabled for a user at a given time. This combination of services might take a number of different forms. For example, similar or same services from different service-domains might be combined, or services in one service-domain have precedence over similar services in another service-domain. Here, these combinations of available services are composed in an expression using operators to describe what it means for a service to match against the composition.

More precisely, the service-domain D of a set of services for a user U is the geographical area in which the services S' for the user U is available such that $S' \subseteq S$, where S is all the services available in D. So, for example, in our model, let D_i and D_j be two service-domains where $i \neq j$ and let S_a and S_b be two sets of services where $a \neq b$. If S_a and S_b are the sets of services associated with service-domains D_i and D_j respectively, and if the user is within D_i and D_j at this time, then a simple case is that the ambient services available to the user is $S_{D_i} \bigcup S_{D_j}$ (set union of S_a and S_b). So, in this example, for a given user U and a service-domain D_i and D_j, a set of services $S_a \bigcup S_b$ are all the ambient services of the user U (assuming D_i and D_j are mutually exclusive). But this might not be the case, for example, we might have a case where the user is within D_i and D_j, but he/she is only allowed to use services of D_j. This might be due to the constraints imposed by the service-provider of the service-domain or the user himself/herself has his/her own

preference of what services he/she wants to be enabled at a particular area. Thus, the services available to a user at a particular area are not all the services associated with the service-domains containing that area but, in some cases, only the combination of some of the services associated with the service-domains.

The combinations of sets of services which can be invoked by a user *U* at a given location are represented using expressions. When the user is in that location, he/she can make a request against the composition of sets of services. These composition expressions are formed using the operators presented in [8]. The operators describe the rules for a service to match against a composition.

3 Prototype Implementation

We first present key concepts used in the implementation in Section 3.1. Then, we describe the scenario we implemented in Section 3.2. Thereafter, Section 3.3 describes the architecture of the prototype.

3.1 Key Concepts

We discuss the concepts of logical area, user types, mapping tables, and service matching mechanisms.

Logical Area. In [8], a user U can be any individual in a certain geographical area and every user in that area can access the same set of services. The geographical area which a user is located in can be defined as a logical area. A logical area can be a small grain area such as a few square feet or an infinite area such as outside the square feet's area, a granularity dependent on the accuracy of the positioning technology. We assume for simplicity here, that given any service-domain D, a logical area L is either completely contained in D or completely outside D, and not partially inside (or outside) D. The location of the user can be represented using a coordinate system or symbolically. In this paper, the location of the user is represented symbolically by the area that the person is in, which we call the user's logical area. Using this logical area, we can determine what service-domains the user is in. If a service-domain D contains the logical area L of the user, then the user must be within D. If a service-domain D is contained in L, then the user is assumed to be not in D, since we cannot ascertain that the user is in D (i.e. the user might be in L but still outside D).

User Type. When implementing applications, users in the same logical area may not have the same privileges to ambient services. Users who have the same privilege to use the services are categorized into user types (e.g. type A). Thus, ambient services for users may vary from one user type to another, and in such case there is not only the user's location but also the user's type that has an impact on what services would be available for that user: a user's user type determines what services will be available for the user in a given-service domain. Two types of impacts are possible which are:

(1) the impact of user type on services in a service-domain, and (2) the impact of user type on the composition of service-domains in each geographical area.
The first impact occurs when a service-provider of the service-domain offers different sets of services to each user type. For example, two users, X and Y are in the same service-domain and suppose the set of services associated with this service-domain is S, then services for X is $S' \subseteq S$ and for Y is $S'' \subseteq S$, where it is possible that $S' \neq S''$ since X and Y are of different user types. In a Library Service Domain, the administrator can add and delete books but the student can only access the book details. The second impact occurs when the service-provider of a given geographical area provides different combinations (composition) of services to each user type, allowing users to have more relevant and tailored sets of services. For example, if a lecturer is in his/her office which is within the campus service-domain, the printing service in his/her office is set to override other printing services in the campus service-domain, but if a student is in the office, the printing service in the campus service-domain is still the only accessible printing service.

Mapping Tables. To explicitly model such impacts, the mapping tables are designed to define: (1) the set of services available in each service-domain for each user type, (2) the service-domains containing the logical area for each logical area considered, and (3) how the sets of services from the containing service-domains are composed for each logical area.
A positioning technology can work out which logical area the user is currently in (call this L), and using (2), the system can work out which service-domain(s) contain L, and so work out which service-domains the user is currently in. If the service-domains are say D1, D2 and D3, and their associated service sets are S1, S2 and S3, then the services available for this user from each service-domain will be $S_1' \subseteq S_1$, $S_2' \subseteq S_2$ and $S_3' \subseteq S_3$. S_1', S_2' and S_3' are determined by looking up table (1). The actual services available for the user will be some combination of S_1', S_2' and S_3' (e.g. $((S_1' \bigcup S_2') \lhd S_3')$). The system can determine what combination by looking up table (3). The mapping tables for (1) and (3) are explained in detail later. Details of (2) are embedded in (3) and do not correspond to an explicit table.

Semantics for Service Matching. Let D be a service-domain and let S_i be its services where *i = 1…n*. Let N_i be the service name of S_i and let T_i be the type of service S_i. For example, when an individual is within a service domain D, a set of services available to the user in service-domain D is to the user in service-domain D is $\{S_1(N_1, T_1), \ldots, S_m(N_m, T_m)\}$, for some m less then n.

When describing the operators, we have assumed that the same matching mechanism is used for every operator. In this section, different matching mechanisms are applied to different operations and are described as follows.

Service Matching for Union (denoted by "$\cup$"). A service S_i matches against S_j, if and only if, S_i's name matches against S_j's name and S_i's type matches against the service type of S_j. Thus, $S_i(N_i, T_i) = S_j(N_{j,} T_j)$ if $N_i = N_j$ and $T_i = T_j$ where $i \neq j$. In forming a union, duplicates are eliminated and since two services are considered duplicated only when their names and types match, this definition tends to lead to larger unions. If we had matched only on type, more services will be considered duplicated, and the unions will tend to be smaller.

Service Matching for Intersection (denoted by "n"). A service S_i matches against S_j, if and only if, the type of S_i matches against the type of S_j. Thus, $S_i(N_i, T_i) = S_j(N_{j,} T_j)$ if $T_i = T_j$ where $i \neq j.$

Service Matching for Restriction (denoted by "\|"). A service r matches against a composition $S_i \mid S_j$ if and only if, r matches the services in S_i, but does not match against services in S_j. In the Intersection operation, a service S_i matches against another service S_j, if and only if, the type of S_i matches against the type of S_i. Thus, similar to intersection, in restriction, we have $S_i(N_i, T_i) = S_j(N_{j,} T_j)$ if $T_i = T_j$ where $i \neq j.$

Service Matching for Overriding (denoted by "$\lhd$"). Overriding is defined using the other operators as shown. Thus, the same service matching mechanisms for Restriction and Union can be used to match services for the overriding operations.

The service matching mechanisms allows the calculation of services with attributes (in our case, assumed embedded in the service names) more sensibly. For example, a service with name "Print to printer A, B, C and D" in service-domain D_i of type "Printing Service" can be overridden by a service with name "Print to printer O" of type "Printing Service" in service-domain D_j by using the above service matching mechanisms since these two services are of the same type (though different names). Union matches using names and types so that in a union, services of the same type, as long as they have different names, can co-exist. But we give full force to restrictions and intersections for their constraining function, by relaxing the matching, i.e. using only types so as to eliminate services of the same type (even if they have different names).

The service matching mechanisms allows the calculation of services with attributes (in our case, assumed embedded in the service names) more sensibly. For example, a service with name "Print to printer A, B, C and D" in service-domain D_i of type "Printing Service" can be overridden by a service with name "Print to printer O" of

type “Printing Service” in service-domain D_j by using the above service matching mechanisms since these two services are of the same type (though different names). Union matches using names and types so that in a union, services of the same type, as long as they have different names, can co-exist. But we give full force to restrictions and intersections for their constraining function, by relaxing the matching, i.e. using only types so as to eliminate services of the same type (even if they have different names).

3.2 Overview of Illustrative Scenario

We consider a scenario where there is a location-based system for a floor area of a university campus which provides location-based services to users. The services are enabled or disabled only within a particular area and relate to the immediate physical surroundings of the user and users' type. Once a user enters the area (typically a floor within a building), he/she is enabled with locally relevant services and these services will be updated as he/she moves to another area. Three service-domain(s) are considered:

- Office Service-domain (DO)
- Printing Service-domain (DP)
- Library Service-domain (DL)

Each service-domain has a set of services associated with it. The Office service-domain has schedule-services, Printing service-domain has printing services and Library service-domain has library services associated with it. In addition, three logical areas are defined in the floor of the building to describe the location of the users: Office Logical Area, External Logical Area, and Library Logical Area.

The Office Logical Area is a region within the polygon with non-dotted line shown in Figure 1. The External Logical Area is the region within the polygon with big dotted line. The Library Logical Area is the region within the polygon with small dotted line. The sets of services in DO are enabled within Office Logical Area, i.e. DO corresponds to Office Logical Area. The sets of services in DP are enabled within External Logical Area and DL in Library Logical Area respectively, i.e. DP corresponds to the External Logical Area and DL with the Library Logical Area. As shown in Figure 1, the logical areas are contained in each other. Thus, for example, when the user is in the Office Logical Area, he/she is also in the Office Logical Area, External Logical Area and Library Logical Area at the same time and thus, will be in the service-domains DO, DP, and DL, i.e., all the services associated with DO, DP and DL will be (potentially) enabled for the user. In this scenario, logical areas correspond to service-domains but it need not be generally so in our model.

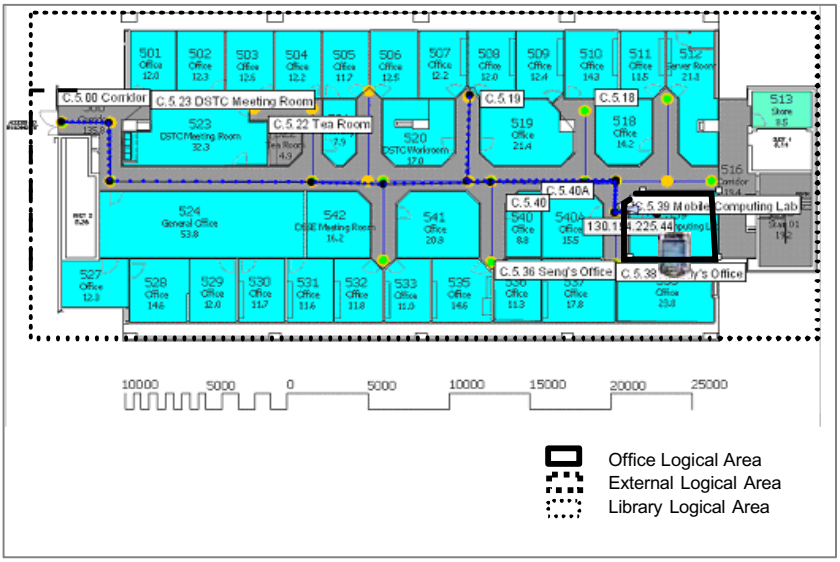


Fig. 1. Floor of a building in the University Campus

However, not all the services in the service-domain are actually available to the user due the constraints imposed by the service-provider. The assumption was made in our system that the service provider is responsible for managing the services for users instead of users having the control over what services they can use. Two types of constraints are imposed by the service-provider: (1) the number of services available to each user for each service-domain is limited based on their user types, and (2) only the compositions of sets of services from service-domains of each logical area are enabled for each user type. For example, similar or same services in different service-domains are unioned, and one service-domain has precedence over similar services in another service-domain.

Mapping Tables for Each User Type. This subsection presents how mapping tables are used to provide the locally relevant and tailored services to users with different privileges to use the services. In our system scenario, three types of users are defined: lecturer, student and administrator. Each user has different sets of services available in each service-domain and is shown in the User Type Matching Tables. In Table 1, the sets of services associated with each service-domain are defined for a Lecturer and Administrator (both have the same privileges).

Table 1. A Mapping Table for sets of services available for Lecturer and Administrator

Lecturer/Administrator

Service-Domain	Set of Services
DO	{View personal schedule, Add schedules, Delete Schedules, Update Schedules, Print to printer O}
DL	{Basic/Advance Search, Renew Loans, View Personal, Information}
DP	{Print to printer A, Print to printer B, Print to printer C, Print to printer D}

The students are allowed to use all the services in DL and DP but none in DO. Sets of services available to students are as shown in Table 2.

Table 2. A Mapping Table for sets of services available for Student

Student

Service-Domain	Set of Services
DO	{ }
DL	{Basic/Advance Search, Renew Loans, View Personal Information}
DP	{Print to printer A, Print to printer B, Print to printer C, Print to printer D}

Mapping Tables for Each Logical Area. The set of services available for a user in a particular logical area will result from evaluating against the composition retrieved from the mapping table. The sets of services available in the Office Logical Area are some of the services available in service-domains DL, DP and DO (since they all contain the Office Logical Area) and the compositions of sets of services for each user type are shown in Table 3. The services associated with DL, DP and DO are denoted by S(DL), S(DP) and S(DO), respectively.

Table 3. Mapping Table for Office Logical Area

Office Logical Area

User	Composition of Sets of Services
Lecturer	S(DL) U (S(DP) ◁ S(DO))
Administrator	S(DL) U (S(DP) U S(DO))
Student	S(DL) U S(DP)

The service-domains containing (or exactly corresponding to) the External Logical Area are DL and DP. The compositions of services available for each user in the External Logical Area are shown in Table 4.

Table 4. Mapping Table for External Logical A rea

External Logical Area

User	Composition of Sets of Services
Lecturer	S(DL) U S(DP)
Administrator	S(DL) U S(DP)
Student	S(DL) U S(DP)

The service-domain containing (or exactly corresponding to) the Library Logical Area is DL. The compositions of services available for each user in Library Area are shown in Table 5.

Table 5. Mapping Table for Library Logical Area

Library Logical Area

User	Composition of Sets of Services
Lecturer	S(DL)
Administrator	S(DL)
Student	S(DL)

3.3 Architecture Overview

This section explains the architecture of the system for proactive discovery and update of ambient services. The architecture consists of four major components: the

Ekahau Positioning Engine (EPE) [4], the Service Calculation Engine (SCE), Mobile Client Application and Web Services.

Ekahau Positioning Engine 2.0 (EPE) is the positioning server which keeps track of mobile users in a wireless LAN. It detects the mobile users who enter a particular geographical location of each user. The user's current location information is then sent to the Service Calculation Engine periodically to calculate which service domain(s) the user is currently in and the composition of services available for each user. The system requires that there is a constant connection between the Ekahau Positioning Engine and the Service Calculation Engine.

The Server Module consists of two main components: Service Calculation Engine and Service Database. Service Calculation Engine (SCE) plays the most important role in discovering and updating of ambient services. Its main task is to calculate the service domains in which the user is currently in and a composition of the sets of services available to the user based on the location information received from EPE. Every time the user moves to a different logical area, a new set of services is calculated and enabled for the user.

The SCE also acts as a server to the client devices which enter a particular service domain. In order to send updated sets of services to the user, it listens to the client device connections continuously. When a user enters the service domain and connects to the server, it establishes a connection with the client device and sends updated sets of services to the client device. As the user moves from one service domain to another, the composition of services available to the user would change and the new set of services are calculated and sent to the client device.

Service Database is a database repository which stores the details of the mapping tables described earlier. It includes: services, service-domains, and service compositions. The service database is on the same host as the Service Calculation Engine. SCE retrieves the services from the service database to calculate the locally relevant services at a given geographical area.

The Mobile Client Application is an application installed in the mobile device of the user. This device is tracked by the EPE. The basic function of the client application is to receive the sets of services from the server and display it to the user. When the user invokes the service, the client makes a request to the web service. The current implementation is specifically aimed for the Pocket PC and is developed using the Microsoft .NET Compact Framework (CF). When the user enters a particular area, they are required to log in with User name, User type and Password, in order to check the user type of the mobile user since different user types have different privileges in the system.

After logging in, once the user clicks the "Connect" button, the mobile device will connect to SCE which is waiting for client connections on the server machine. It then receives the services which are calculated and transmitted by SCE. A new set of services is sent to the mobile client application every time the user moves from one logical area to another (as detected by processing data from Ekahau). Thus, the services displayed to the user are updated every time it receives the new sets of services.

Fig. 2. Services available in Area 1 (External Logical Area) displayed on a user's mobile device

For example, when the user moves from Area 2 (Office Logical Area) to Area 1 (External Logical Area), a new set of services will be enabled for the user. The services available in Area 1 are displayed for the user in his or her mobile device as shown in Figure 2.

4 Conclusion and Future Work

We have investigated a type of location-based service which we called ambient services. Such services, by definition, have geographical boundaries or areas of relevance and usefulness. We called these areas service-domains as they are associated with a set of services. We have also discussed the case of a user being in multiple service-domains at the same time, and how the set of suitable ambient services for the user can be computed. The system utilizes mapping tables and operators to select services from the containing service-domains, in order to assemble a list of services for the user. The prototype is a proof-of-concept, and its evaluation demonstrates the feasibility of ambient services. Our prototype generates the list of services for the user. Beyond this, for the actual invocation of the services, other architectures can be used such as in [9].

Further work will involve improvements to the performance of the prototype, exploring integrative location technologies (e.g., integrate with GPS outdoor positioning technologies for larger service-domains and larger-scale services as noted

in [3,6,12]), and investigating the run-time handling of services (e.g. what to do if a user walks out of the service-domain of an executing service). Other applications can be considered such as location-based messaging [2], reminder services, and device control. Hodes and Katz [5] discussed mechanisms to control devices in the environment from the mobile device. They use base station beacons instead of a location server, do not use Web service standards, and do not organize services based on service-domains. Ambient services can be constructed for their purpose. Our work is similar in spirit to that in [11] where spontaneous interaction with resources occurs by virtue of proximity to resources, but we do not employ predictive schemes. Predictive schemes can be used in our work to improve performance.

References

1. Ahlund, C., Zaslavsky, A. and Matskin, M. Supporting Mobile Business Applications in Hot Spot Areas with Pervasive Infrastructure. In *Proceedings of the 1st International Conference on Mobile Business*, Greece, July 2002.
2. Chang, E. L. Hanging Messages: Using Context-Enabled Messages for Just-In Time Communication. Masters Thesis. Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, 2001.
3. Cheverst, K., Davies, N., Mitchell, K. and Friday, A. (2000). Experiences of Developing and Deploying a Context-Aware Tourist Guide: The GUIDE Project. In Proceedings of Internatinal Conference on Moble Communication (MOBICOM), Boston, MA, USA, 2000. ACM.
4. Ekahau (2002). http://www.ekahau.com.
5. Hodes, T. D. and Katz, R. H. Composable Ad hoc Location-based Services for Heterogeneous Mobile Clients. Wireless Networks, vol 5, pp. 411-427, 1999.
6. Jagoe, A. *Mobile Location Services: The Definitive Guide*, 2003, Prentice Hall.
7. Leeper, D. G. (2001). A Long-Term View of Short-Range Wireless Networking. *IEEE Computer*, 34(6): 39-44.
8. Loke, S.W.. Modelling Service-Providing Location-Based E-Communities and the Impact of User Mobility. In *Proceedings of the 4th International Conference on Distributed Communities on the Web* (DCW 2002), (eds) J. Plaice, P.G. Kropf, P. Schulthess, J. Slonim. Sydney, Australia, April 3-5, 2002, pages 266 - 277, Springer-Verlag, LNCS 2468.
9. Pilioura, A., Tsalgatidou, S., and Hadjiefthymiades. Scenarios of Using Web Services in M-commerce. ACM *SigEcom Exchanges*, Vol. 3, No. 4, January 2003, pp 28-36.
10. Priyantha, N. B., Chakraborty, A. and Balakrishnan, A. The Cricket Location-Support System. In Proceedings of MOBICOM 2000, Boston, MA, 2000, ACM Press.
11. Troel, A., Banatre, M., Couderc, P., AND Weis, F. Predictive Scheme for Approximate Interactions. In *Proceedings of the International Workshop on Smart Appliances and Wearable Computing* (IWSAWC'01), pp. 235 – 239, 2001.
12. Varshney, U., and Vetter, R. Mobile Commerce: Framework, Applications and Networking Support. *Mobile Networks and Applications* 7, 185-198, 2002.

Information Gathering for Dynamic Selection of Web Services

Amir Padovitz, Shonali Krishnaswamy, Seng Wai Loke

School of Computer Science and Software Engineering, Monash University,
Caulfield East, VIC 3145, Australia
{amirp, shonali, swloke}@csse.monash.edu.au

Abstract. Gathering status and QoS information from web services at run time can lead to a more dynamic interaction between clients and web service providers. Such a model provides better reliability for client applications that are dependent on web services for critical data/information. This paper presents three different models based on traditional RPC and mobile agents for gathering information regarding the status and QoS parameters of web services at run time.

1 Introduction

Distributed computing is evolving to a service-oriented model and is leading to the emergence of web services, which aims to change the way we perform communication between disparate and remote applications [1]. The concept of web services offers a flexible and open model, targeting to solve various restrictions and problems of traditional distributed architectures. Interaction between applications in heterogeneous platforms, independent of language and vendor specific middleware will become feasible by using communication mechanisms that adhere to common open standards, which are the basis of web services. By building on existing technologies and conforming to those well-known standards, web services aim to offer a powerful model that can be used to describe, discover and execute remote services by client applications that use these services.

Despite the support of different frameworks for the standards that make up web services, the current model of web services still needs to address many challenges of incorporating web services into more dynamic environments [2, 1] and addressing issues such as reliability for applications that use external web services based on the availability and quality of service of web services.

To ensure features such as reliability of client applications, it is imperative for the client to be aware of changing conditions at the service providers at runtime. These features may influence the viability of a web service for the application.

Characteristics such as response time of web services (i.e. time to complete the service for a client request), availability of service providers, prices of web services and quality of service provided can change at runtime, making a specific web service

either more or less suitable for the client. Monitoring these parameters and introducing this information to the client application can improve the overall performance of the application and help create a more robust and reliable application. By introducing runtime information concerning external web services, client applications that use web services will be able to dynamically alter their behaviour at runtime. The client abilities to cope with changing factors in the environment and as a result optimise its execution both from the software engineering aspect (e.g. response time and availability) and the business aspect (e.g. price) will become feasible.

The paper is organized as follows. In Section 2 we present architectural models to support the dynamic selection of web services. Section 3 presents the design of a prototype implementation. We conclude and discuss future work in section 4.

2 Architectural Models for Runtime Information Gathering

We propose an infrastructure, shared by consumers and service providers that will perform collection of information, related to availability and QoS parameters of the appropriate web services at runtime. The type of information to be collected is dynamic and can only be discovered during the application execution (e.g. estimated response time of a web service, price, etc.). Such information can affect the clients' decision on web service invocations and ensure the reliability of the client's application by activating only available and more reliable (a QoS parameter) web services. The information gathered can also improve the overall client application performance by choosing to activate web services with fast completion times as well as improve other business oriented aspects of the application, such as selecting web services in specific price range or with specific quality of service.

Figures 1 illustrate client activation decisions based on information gathered at runtime from the service providers according to the client constraints. In figure 1, the client is concerned with availability and response times of a web service. After retrieving related information from the service providers, the client activates the fastest available web service. This behaviour contributes to the robustness of the client application, by avoiding activation of unavailable web service and selecting the fastest available.

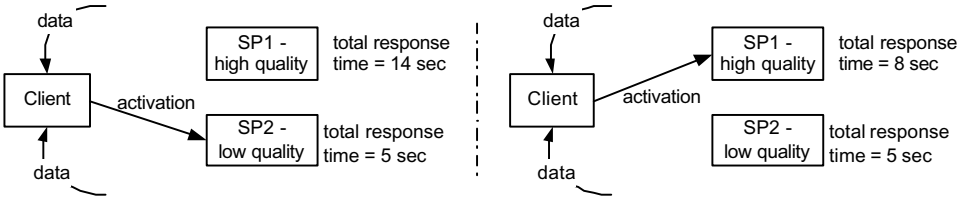


Fig. 1. Web services activation according to response times and availability

We propose three different models for gathering information for dynamic selection of web services. Each of these models has strengths and limitations and is best suited for particular situations. We aim to integrate all three approaches and create a model that caters for different situations.

2.1 RPC Based Model

In traditional wide network scenarios (e.g. the Internet) the most straightforward approach for gathering information for web services activation would be to use RPC (remote procedure calls) for communicating information between hosts in the network. For platform independence, web services themselves can be the means by which communication between two hosts is performed. Information could then be easily and generically sent and received between all the participants in the network. This kind of implementation is generally beneficial in wired networks, as multiple connections need to be handled, which may become difficult in wireless environments where connections are less reliable. In this model a component-oriented approach is taken, in which a client/service provider is treated as a black box. When a service provider receives a request, it may become a client and actively request information from other service providers. This approach simplifies the programming complexity of an environment consisting of many service providers.

In this architecture, decisions regarding the ways to query deeper nodes in the network or decisions on which nodes to query and when, are all delegated to the service provider, which then becomes a client of the system. In this component-oriented scheme, the initial client delegates future decisions and implementation details to subcontractors in the form of the service providers.

In the RPC model, once an RPC request is launched to a certain service provider, parameters contained in that request cannot be changed. The service provider may retrieve information from other service providers according to the initial parameters specified by the client. The initial client no longer has control over the operation and cannot change parameters or conditions for that search of information. Another problem is the high number of messages sent between client and service providers. This limits the reliability of such a system in a wireless environment where connections are less reliable. Furthermore, the higher the number of participants, the more messaging is involved, resulting in more traffic congestion.

2.2 Mobile Agents Based Model

To address the issue of wireless connectivity and client control we propose a second model based on mobile agents. We examine two approaches that differ in the type of behaviour agents perform. In the first model, agents are launched by the client, arrive at the service providers, query information, and if needed, continue to look for dependent services for required information. Agents contain client restrictions such as timeouts and maximum number of hops as well as other data that pertain only to the specific client. Following this approach, two agents of different clients may act differ-

ently under similar circumstances given different client directions for behaviour. Figure 2 shows a possible strategy performed by agents a and b that are launched by the client. Upon arriving at a service of type A they are redirected to retrieve information from service providers B1 and B2. Then they are redirected again to gather information pertaining to services C1, C2 and C3. After arriving at B2 agent b clones itself into 2 agents, each travelling on a different path to accomplish its task.

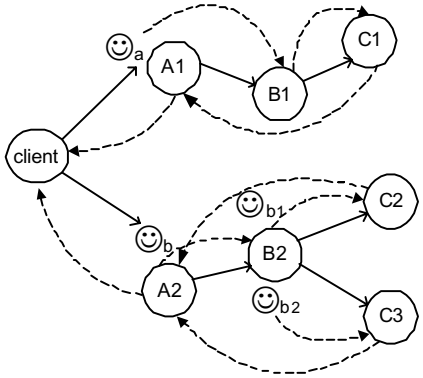


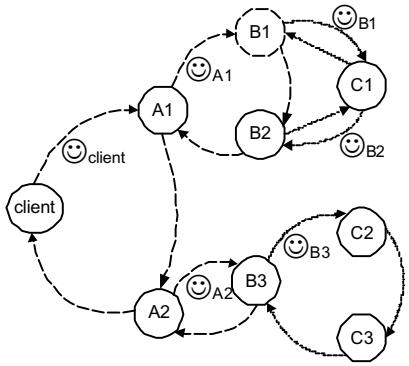
Fig. 2. – First mobile agent model

In this approach we gain client control of the agent behaviour in deeper node levels, after the initial encounter with the first service provider. In addition, we also gain better reliability in wireless environments, as the amount of connections is highly reduced compared with the RPC model. In this model the client (a wireless device) only maintains connections as per the number of agents it initially launches. A mobile agent should also be embedded with the ability to change its migration path if it encounters disconnected or unresponsive nodes. Despite the higher level of abstraction for programming mobile agents, implementing agents to work in a large network of dependable service providers could pose complexity. Instead of treating the service providers as black boxes as in the first model, the agents need to be programmed to move around the network autonomously and respond to possible changes in the environment.

The main disadvantage of this approach however is that it is less realistic from a business perspective. It is unlikely that a client agent would be permitted to be redirected and interact with nodes that the service provider is concerned with. Service providers may have for example private agreements with other service providers and would not want to send an agent that represents the initial client. It may also be against the best interest of the client, since sometimes a direct request for a service from the client would result in a higher price than if the service provider had handled it.

The second approach assumes interaction of the client agent only with the required first level service providers. This approach is similar to the RPC one, where service providers are treated as “black boxes”. Although client control is lost, this model still maintains better performance in wireless environments, compared with the RPC model.

Figure 3 shows this approach; agents operate on behalf of their clients and are re-



stricted to interact only with the required first level service providers.

Fig. 3. – Second mobile agent model

The differences between the RPC and Mobile Agents models in terms of the number of wireless connections are shown in figure 4.

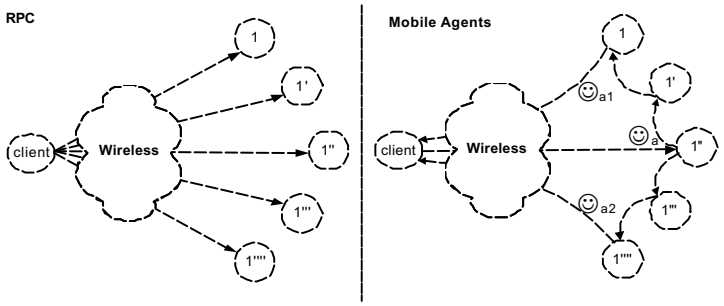


Fig. 4. – Differences in number of wireless connections between the models

In the RPC model the client manages at least five connections to all five service-providers, while in the Mobile Agents model only one connection is established and used to send a single mobile agent that clones itself and which arrives at all the service providers autonomously.

2.3 Circulating Mobile Agents Based Model

The two previous models are suitable to work with web services that are expensive to purchase and/or consume an overall long processing time. In such cases clients may be willing to wait for results if it would translate into significant reduction in money that is spent or if the additional time spent on the selection of the most appropriate web service is not significant, compared with the activation of the web services

themselves. Since the task of collecting information is time consuming, it is less likely these approaches will be utilized when it is imperative to perform fast activation of short processing time and inexpensive web services. For such scenarios, a third model, the Circulating Mobile Agents, shown in figure 5, is proposed that can provide service providers related information "on demand". The idea behind this model is having mobile agents periodically circulate the path of the service providers and retrieve information. The information is given to the client, who then performs web service activation based on the latest information that is available. In this scenario, information that arrives is more up to date and is available sooner.

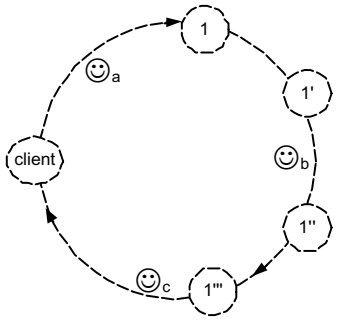


Fig. 5. – Circulating mobile agents

The limitations of this model may also suffer from redundancy. Depending on the client application, agents may circulate the network, retrieving information without any current need to do so. To minimize this redundancy, the ability to control the amount of circulating agents and the duration of their life cycle need to be introduced.

Table 1 and table 2 summarize client considerations as to which model would be advantageous under different circumstances.

Table 1. – Activation considerations according to service characteristics

Service Characteristics	RPC	MA	Circulating MA
Expensive, long processing time	X	X	
Fast response is important			X

Table 2. – Activation considerations according to network characteristics

Network Characteristics	RPC	MA	Circulating MA
Wired	X		
Wireless		X	X (depends on number of agents)

It can be seen that the Circulating Mobile Agents model is most appropriate when clients need fast results of information gathered. The RPC model is more suitable to be

activated in a wired network, whereas the Mobile Agent and Circulating mobile Agents models may be more appropriate to use in wireless environments.

3 Implementation of Prototype

We now present the implementation of a prototype that enables us to perform experimental evaluation of the three models.

The main functionality is implemented in the *WSAdvisor* components; client applications interact with this functionality either directly - when requesting web service activation recommendations, or indirectly – when updating information on new possible web services in the repository. *WSAdvisor* components query that repository to obtain information on web services and create itinerary for communicating with the service providers.

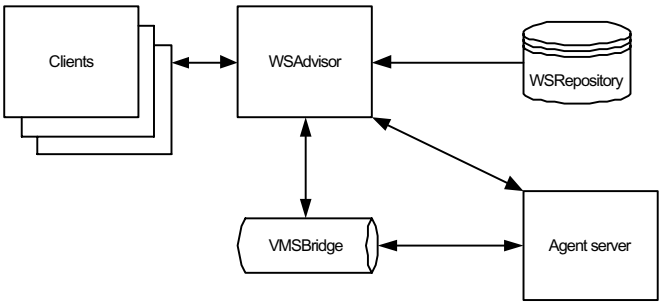


Fig. 6. – High-level overview of the system prototype

An agent server is used to launch new agents with itineraries to service providers' destinations. A communication utility object – *VMSBridge* is used to facilitate decoupled communication between the main *WSAdvisor* components and other third party applications. *VMSBridge* also serve as a link between different platforms. Since many current usage and implementations of web services are developed and run on Microsoft .NET and most mobile agent toolkits, and in particular IBM's Aglets (which is used in the current implementation) operate in a Java virtual machine, the system needs to be able to work with both environments at the same time. Figure 6 draws a high level overview of the system, figure 7 shows GUI screen of a client application using the *WSAdvisor* system to select a suitable web service.

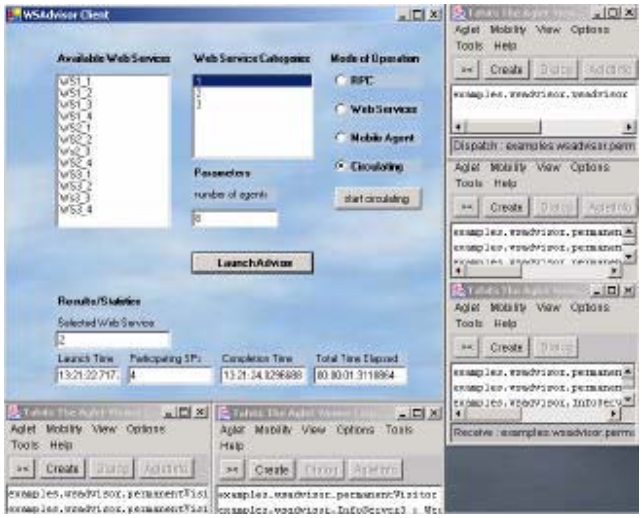


Fig. 7. – Client GUI using the *WSAdvisor* system

4 Conclusion

We have investigated an architecture that promotes performance and robustness of web service consumer applications by obtaining availability, quality of service and other business related parameters on web services, which are controlled by or accessible to service providers, at runtime. The collection of such information is performed according to a set of constraints, provided by the client's application, and the most suitable web service at the moment in time is chosen according to the client requirements. Using such architecture may reduce some of the problems associated with web services, such as the need to ensure the reliability and availability of external web services, the need to obtain information of service providers' business information and reduce the overall activation time of web services by consumers. As a result, clients' applications can become more robust and improve their overall performance. Clients would also gain more control over external resources (e.g. web services) and would improve business related decision-making and performance. Three models for runtime gathering information were proposed implemented with a prototype implementation.

References

1. Hailpern B., Tarr P., Software Engineering for Web Services: A Focus on Separation of Concerns, IBM Thomas J. Watson Research Center. http://citeseer.nj.nec.com/477241.html

2. Glass G., The Web services (r)evolution, IBM developerWorks, November 2000.
http://www-106.ibm.com/developerworks/webservices/library/ws-peer1.html?dwzone=ws
3. Glass G., The Evolution of Web Services , Prentice Hall, June 21, 2002.
http://www-106.ibm.com/developerworks/library/ws-peer1.html?dwzone=components#N40003F

.

A Taxonomy of Information Technology Services: Web Services as IT Services

Andrea Stern, Joseph Davis

School of Information Technologies,
University of Sydney, NSW 2006 AUSTRALIA
Tel. +61 2 9036 9108

Abstract. This paper examines the web services model in the context of models of a broader set of information technology (IT) services. It compares and contrasts the web services model with these models and proposes an approach to describing and classifying IT services on the basis of the complexity (as reflected in the structure and uncertainty) of the wider range of services. It also explores some critical implications of such a taxonomy, including a discussion of issues relating to managing and contracting for IT services which need to be comprehensively addressed in the service- oriented computing paradigm.

1 Introduction

Web services increasingly attract both popular and research interest. They are novel in their techniques of automated discovery and orchestration and in their dynamic integration of standardised application modules. Their XML-standard based communication techniques and underlying service oriented architecture (SOA) give organisations the potential to increase flexibility and agility and represent a radical departure from traditional monolithic custom-developed applications and specialised client interfaces. The standardisation embodied in the web services model structures the environment in which they operate, and is a key to their interoperability.

A challenge for the development of web services however, is whether the model can be extended to encompass more complex areas of IT service provision in less structured

environments. For such services, conditions of service are negotiated and customised, outcomes are difficult to control or predict, services may be linked in hierarchies of related services and complex service management and maintenance issues may arise.

An approach to this challenge is to see Web services as a specific category of what could be broadly referred to as IT services. This broader class of IT service, like web services, is characterised by a service view of IT in contrast to the application, activity, or product view. Such a view shifts the focus from managing ownership of IT infrastructure and applications to managing negotiation and delivery of effective IT services. By contextualising web services in this way and examining the relevance to web services of the different models of IT services that emerge, it may be possible to inform and enhance the development of the web services model to tackle other areas of more complex endeavours in IT services.

This paper reviews IT service models to be found in the literature on application service provision, network services management and IT service management and examines their characteristics. It then compares and contrasts web services with the broader range of services, on the basis of differences in degree of structure and uncertainty. It argues that degree of structure and uncertainty indicates the complexity of the models represented by the services. On the basis of those comparisons, it argues that IT service models can be classified by complexity and that the resulting taxonomy can be useful in identifying issues relevant to web services. Some implications of this for web services and service oriented computing (SOC) are explored, including the relevance of techniques for negotiation and managing IT services.

2 Web services

Web services are internet-based interoperable IT services proscribed by a formal model which support business functions within the SOC paradigm. Because they are standardised, self describing and XML based, they can be discovered and invoked automatically, are dynamically bound at the time of use and are interoperable in multiple environments. According to a report by the Stencil Group [1], the modularisation of applications fundamental to web services, allows a flexibility and agility whereby organisations can add new offerings or reorient existing functions. Through modularising applications and making the modules interoperable through standard protocols on a fee-for service basis, web services reduce the need for customisation and up-front investment, lower the transaction cost of commonly needed IT services, and offer organisations flexibility and agility in it integrating and enhancing their IT-based functions. “Fundamental to web services, then, is the notion that everything is a service, publishing an API for use by other services on the network and encapsulating implementation details” [2, p2.].

Taxonomies too play an important role in the flexibility and versatility of web services. They provide searchers of the Universal Description and Discovery (UDDI) layer ever greater precision in specifying business-relevant criteria through the availability of multiple taxonomies and identifier systems [3].

Three important foundations underpin the flexibility of web services. Firstly, the layered architecture of SOC provides a standard framework for the dynamic integration of web services. Secondly, just as standardisation and modularisation of processes has improved efficiency and versatility in the manufacturing sector, so the modularisation and standardisation of services and processes in SOA contributes to the efficiency and flexibility of web services. Lastly, the use of XML as the standard language for message passing, frees web services from depending on conformity to a single operating system or other proprietary software for integration. XML contributes significantly to the flexibility and extendibility of web services, allowing for example, web service publishers and searchers to specify business-relevant criteria with an increasing range and precision through the ability of the UDDI layer to operate with multiple taxonomies.

But while XML and the web services protocol stack underpin the flexibility inherent in web services, the services operate with constraints in the three important areas: negotiating variable conditions of service, managing relationships among services in a hierarchy of services and providing support for a service over time.

The very standardisation which contributes to the flexibility of web services also restricts services to those for which the conditions do not need to be negotiated, or where the negotiations have taken place outside the domain of the web service and are implied in the service. For example the UDDI enables technical interoperability but does not allow for the description of the policies or business rules governing the interaction. Cerami [4] comments that there are no mechanisms for automating business relationships pertaining to e.g. pricing or delivery schedules, for negotiating conditions of service, for monitoring reliability and availability, for the legal ramifications if the service delivery is not made, and that neither can assumptions be made that a service is bug-free and available at guaranteed times.

The web services model, while technically advanced and highly standardised, is limited to services which can be contracted and used automatically as standardised commodities, with relatively simple transactions. The precisely defined structure of the service does not appear to allow for the complexity of negotiations of conditions of service, or for changes over time. Nor does it provide for a hierarchy of sub-services in a chain of services. For example, the modularity and integrative ability of web services gives them a versatility which lends them ideally to support the process-oriented view of an organisational value chain. However, while SOC enables end-to-end processes and services to be automated and integrated as a series of entities, the issue of managing relationships among entities

does not appear to be addressed at other than the technical level. The automation of a series of simple processes such as order entry via web services is a different task to the automation of the comp lex composite processes of an order-to-fulfilment supply chain.

Services not only need to be negotiated and delivered but supported over time in alignment with contracted conditions of service. Processes for managing and evaluating the ongoing service such as availability, capacity, problem management and contingency might be seen as an integral part of a service. Some possible future developments in web services in this direction are suggested by Arsanjani, Hailpern, Martin, & Tarr [5]. They suggest, for example, that future developments in object oriented web services technology, in the direction of e-utilities, will increase the sophistication of operation of web services and permit control over performance, reliability, metering and level of service. This could effectively enable services to be dynamically controlled by service level agreements (SLAs) with conformance automatically verified. In this view, SLAs are treated as a component in the building and deployment of services. Similarly, Dan et al. [6], see the formal specification of SLAs as an integral part of the Web stack for the differentiation of services. In other work, XML-based SLA languages are proposed for dynamic electronic services which could enable the automated management, monitoring and enforcement of quality of service guarantees by service providers [7, 8].

3 Other IT service models

A closer examination of other models of IT services in relation to web services might indicate a direction for further developments of web services. The following section views the development of other IT service models, analyses their characteristics and compares and classifies these services. The areas chosen for analysis represent the main areas of IT in which concepts of IT services are discussed in the literature, namely, application service provision, IT service management and network management.

3.1 An emergent view of IT as a service

A service view of IT sees IT-related activities as a set of processes which are carried out by a provider to deliver a service to a consumer, using IT and related resource components, in accordance with a negotiated agreement and maintained and managed over time taking into account failure and change[1]. It emerges in the literature on

[1] Our paper takes the view that IT systems provided from within an organisation can be described as services in the same way as IT systems sourced from outside the organisation, i.e., from the market. This concept is based on the economist, Oliver

application service provision, network service provision and IT service management discussed in the following sections.

Discussion in these domains expose an end-to-end view of a service, i.e. they take an integrated perspective of all the components of a service (processes and infrastructure), in their relationship to each other. For example, Park, Baek, & Honk [10] suggest that rather than looking at network management or systems management in isolation, this view looks at a service from the perspective of the client. This gives a holistic view of resource usage, so that, for example, the availability of a single component is only relevant within the context of the availability of all the other components in the supply chain which delivers the service. Similarly, Koch [11] suggests that the uptime of a network router is of no interest to the business user of the IT service in isolation. Rather, the availability of the entire service is, as are the cost of the service and the definition of the service in business terms. An end-to-end service may also consist of a hierarchy of services supplied from one domain of control to another but the series of related services may be viewed as a single service for contractual and other administrative purposes. Treatment of the related services as a single service is dependent on there being techniques and processes for managing the relationships among the components of the services in the chain. Methodologies for managing network and other IT services also focus on the roles of the provider and consumer, the service level agreement and the support processes for the service lifecycle.

3.2 Application service providers

The idea of delivering a service as opposed to a product is fundamental to the operation of application service providers (Salesforce.com, for example) In providing applications for the use of clients via the internet on the basis of a fee for usage, clearly what is being delivered is not a product but a service. According to Patnayakuni & Seth [12], the ASP model essentially converts a software product into a software delivery service and is a paradigm shift in application delivery. ASPs are responsible for the functionality, availability and reliability of a hierarchy of underpinning components and sub-services (software, hardware, expertise, telecom providers and hardware suppliers). Just as the ASP negotiates and articulates the conditions of its service provision with its clients, so too, it must negotiate conditions with its own underpinning service providers and the ASP

Williamson's, description of the parallels between contracting goods and services in the market and producing them within the firm, in that they are "alternative instruments for completing a related set of transactions" 9. Williamson, O.E., *Markets and hierarchies, analysis and antitrust implications : a study in the economics of internal organization*. 1975, New York: Free Press. xvii, 286.

itself becomes a service provider in a chain or network of service providers. The ASP itself can also be an underpinning service to a business process outsourcer (BPO), which contracts with customers to carry out the actual business processes on behalf of the customer.

Related to ASPs also, is the introduction of software delivery as a service. Greschler and Mangan [13] describe Software as a Service (SaaS), as a concept promoted by Microsoft, Oracle, Sun and others as a central seamless way to deliver and administer software, based on user subscription and payment per use, rather in the manner of a utility service, such as water and electricity. In this paradigm, rather than software being acquired and installed locally as a product, the software application is remotely hosted and streamed on demand to the receiving computer where it is executed without installation.

ASPs and SaaS implicitly represent a model of services, which incorporate a provider, a consumer and agreements about the service and hierarchies of services, although a specific model is not discussed in this selection of the literature.

3.3 IT service management

The field of IT service management (ITSM) has a model of services which includes standardised service management processes, such as SLA negotiation. The model is driven by a customer focused end-to-end view of IT services and is not limited to any particular type of service or environment. It encompasses IT services from traditional in-house IT development and operations to e-services. While standards have been described for the model at high level it has not been standardised at a detailed implementation level.

ITSM methodologies are concerned with managing the provision of IT services from one organisation or division to another. The provision of services includes both the on-going delivery of the service and the support of it. The methodologies take a customer-focused, high level business perspective, viewing the information systems delivered by the IS/IT division as IT services defined by, and driven by, business goals i.e. “a set of related functions provided from the IT infrastructure in support of one or more business areas perceived by the customers as a coherent and self contained entity” [14: p. 6]. Lewis and Ray [15] use as an example of a service, an investment firm deploying Web servers, network infrastructure and application software to allow customers to trade stocks with their Web browsers. This business function then might be labelled Web-based stock trading and the IT service supporting this might be called the stock trading support service.

There are a number of methodologies widely used for managing such IT services. They are based on a concept of ITSM as “a set of processes that cooperate to ensure the quality of live IT services, according to the levels of service agreed with the customer” [16: p iv].

ITSM inter-related processes are designed to deliver and support IT services. They aim to ensure that agreements represent a shared understanding of both customer and provider in supporting business needs throughout the lifecycle of the service as circumstances and requirements change. The processes include cost, change, and problem management, as well as availability, contingency and capacity management. A service level management process negotiates with its customers with regards to the services to be delivered and the conditions under which they are to be delivered, and a configuration management process controls the resource components (including sub-services), on which the delivery and support of the service depends.

ITSM methodologies include ITIL, (the Information Technology Infrastructure Library), a public domain framework for ITSM which is the foundation of most subsequent ITSM models; IBM's ITPM (Information Technology Process Model); Microsoft's MOF (Microsoft Operating Framework); and ISM (Integrated Service Model). The coverage of the methodologies varies. ITIL limits the definition of IT services to the operational or deployment stage of the system lifecycle including the operation, management, maintenance and enhancement of applications and infrastructure. Other, related, methodologies put IT services in a broader context, and include the development and procurement of applications and infrastructure. The ISPL (Information Services Procurement Library), described by Dekker and Hendriks [17], for example, is a set of practices codified as procedures for managing the processes of procuring IT services, whether they be software development services (projects), or operational services,. The methodologies also include specifications for managing information about all the resource components in the IT infrastructure (e.g. hardware, software, SLAs, contracts and sub-services, licences, skills) which are used in the delivery of the service. Mapping these components to services in which they are deployed, however is not a core part of the methodologies although there are instances where it is implemented.

In this model, services are described in a standardised way, i.e. by service purpose and function, scope, cost, service levels (availability, capacity, contingency), support provisions, evaluation methods and underpinning services (sub-services). The model encompasses IT services in general, regardless of scale or complexity, but it has not generated standardised implementation methodologies or automated management processes.

3.4 Network service management

A model of end-to-end services can also be found in literature on the provision and management of network services. It arises in discussions of instruments proposed for automatically evaluating the compliance of services supplied with agreed service parameters. For example, Park and Baek [10] propose a utility model for capturing the management and control aspects of SLAs for multimedia Internet services. Managing and

charging for end-to-end services requires the infrastructure components involved in the delivery of the service to be mapped to that service, even though the components may be controlled in multiple domains. Bhoj, Singhal, & Chutani [18], describe models of compliance evaluation agents which operate across multiple domains in this way. More recent work proposes a generic model of services for the network services environment, to help meet the need for user- and quality-oriented service management [19].

Reflecting a user-centric rather than a device-centric approach to service management in the network services field, is the service model which Garschhammer et al. [20] describe as being designed to support service planning, provisioning and operations as well as service management at the customer interface, The model is intended to "help to analyse, identify and structure the necessary actors and the corresponding inter- and intra-organisational associations between these actors" [20: p.1]. It also accounts for hierarchies of services and maps to that service all the resources required for it, as well as any variables which may impact its quality.

The model is a conceptual meta-model, which can be operationalised to provide both a customer view of the service as well as the provider's view of the realization of the service and uses a case-driven methodology for applying the model. The methodology also accounts for dependency relationships between a service and the resources required to realise the service including knowledge, staff, software and hardware and sub-services, thus enabling the modelling of provider chains. The model is further enhanced by a specification for mapping device-oriented quality of service parameters to user-oriented services. It includes too, the specification of a language for the description of the calculation metric and allows for the inclusion of aggregation of metrics from both devices and sub-services on which the service is dependent [19]. In this way, it addresses what is recognised as a crucial problem in service level management, which is the mapping of low level performance information to high level service parameters [21].

The model is similar to the ITSM model both in that it is customer centric and in the view it takes of the infrastructure resources on which a service depends – i.e. knowledge, staff, software and hardware. Unlike the ITSM model, however, its scope of focus has been the particular domain of network services, not IT services in general. It has been operationalised to provide implementation methodologies to identify, aggregate, manage and monitor the resources on which a network service depends, but does not incorporate the inter-related set of organisational processes for managing the delivery and support of a service through its lifecycle of the ITSM methodologies.

4 Comparison of aspects of service models

The models of IT services discussed above can be summarised and compared in a number of ways, highlighting significant similarities and differences among them.

For example, each model refers, either implicitly or explicitly, to essential elements which define a service in that model. The following table summarises these core elements across the range of models. Reflecting also the recent work in the automation of techniques in service delivery and management, the summary indicates for each model which elements are subject to automated techniques.

Table 1. Summary of core service elements by model

	ASP		**ITSM**		**Network services**		**Web services**	
Service element	Pres.	Auto.	Pres.	Auto.	Pres.	Auto.	Pres.	Auto.
Service content description	x	-	x	-	x	-	x	x
Provider	x	-	x	-	x	-	x	x
Consumer	x	-	x	-	x	-	x	x
Customised SLA for conditions of service	x	-	x	-	x	-	-	-
Resource mapping	-	-	x	-	x	x	-	-
Service hierarchies	-	-	x	-	x	x	-	-
Lifecycle support (Failure /change)	-	-	x	-	x	-	-	-

From this summary, it appears that the models are differentiated in two obvious ways. Firstly in the range of elements they encompass and secondly in the degree of automation employed in the model. The summary also shows that the least automated service model

encompasses the widest range of core elements while the most automated model contained the fewest core elements.

To explore these differences further, we summarised the models by characteristics, other than the core elements, revealed in their descriptions. Those characteristics are: domain of services addressed by the model, degree of structure of those services and level of uncertainty of their operational environment. We chose those characteristics because the description of the services in the literature revealed differences in them sufficient to warrant further investigation into the implications for understanding relationships among the models.

"Degree of structure" refers to the inherent structure in the task being undertaken by the service. This includes the degree to which the specifications being met by the service are clear, the degree to which the impact of the service is clear, and the degree to which the processes being automated by the service are able to be repeated, routinised and standardised. For example, the degree of structure of a service model could be considered high when the level of standardisation in a service enables those standards to be expressed and manipulated computationally, as in web services. This reflects the relationship drawn by H.A. Simon in the field of decision support systems, "between problem solving-strategies and the nature of the task" [22, p.67], in that a standardised and automated problem solving strategy can be applied to a task for which the rules are clear or structured. An unstructured task does not allow routinisation or automation. It follows then that the degree of structure required for computational expression might restrict the applicability of a model such as the web services model to highly structured environments.

"Level of uncertainty" refers to the extent of predictability of contingencies over time. The contingencies might relate to IT components, organisational components, providers or consumers. For example, customisable services with individual SLAs reflecting complex human and organisational issues, or complex technology, which impact the delivery over time, would have a high level of uncertainty because the conditions of the service are not controllable or predictable by the provider. Services which are not negotiable and are strictly controlled by the provider would have a much lower degree of uncertainty.

The table below (table 2), summarises these characteristics of the service models and enables us to compare them on the basis of the service domain, the degree of structure and the level of uncertainty.

Table 2. Summary of service characteristics by model

	ASP	ITSM	Network services	Web Services
Service domain	Implicit service model of common but customisable services.	High level model of customisable generic IT services.	Specific to network services.	Restricted to services describable and discoverable electronically.
Degree of structure	Moderate degree of structure: limited by delivery mode and need for economies of scale.	Low degree of structure: deals with a wide range of organisational and human factors in the negotiation, customisation and management of services through standardised operational processes	High degree of structure: standardised and automated processes.	High degree of structure: standardised, automated processes, procedures and language, including taxonomies.
Level of uncertainty	Moderate level of uncertainty: provider controls some environmental factors.	High level of uncertainty: individual services defined by negotiated SLAs and impacted by org. factors over time.	Moderate level of uncertainty: model is specific to one kind of service environment.	Low level of uncertainty: the service and conditions are controlled by the provider.

The summary suggests that it may be useful to classify IT service models on the basis of the degree of uncertainty and structure of the IT services they represent.

5 Classification of service models by complexity

If we express complexity as a function of degree of structure and level of uncertainty, then we could say that IT services with a high degree of complexity are those which operate in an environment of a high degree of uncertainty and a low degree of structure. Conversely, IT services of a low degree of complexity are those which operate with low uncertainty and high structure. We can then classify services according to this taxonomy.

In classifying service models in this way, the web services model could be considered to have a low degree of complexity because it is structured, standardised and has relatively knowable and manageable contingencies. The ITSM model would have a high degree of complexity because by embracing customisable agreements on conditions of service it accommodates the possibility of unknowable contingencies and a relatively low degree of structure. The network services model would be of moderate complexity as it could operate on a large scale but in an environment where requirements and contingencies are relatively knowable and stable. The diagram below (figure 1), shows a classification of the service models on this basis.

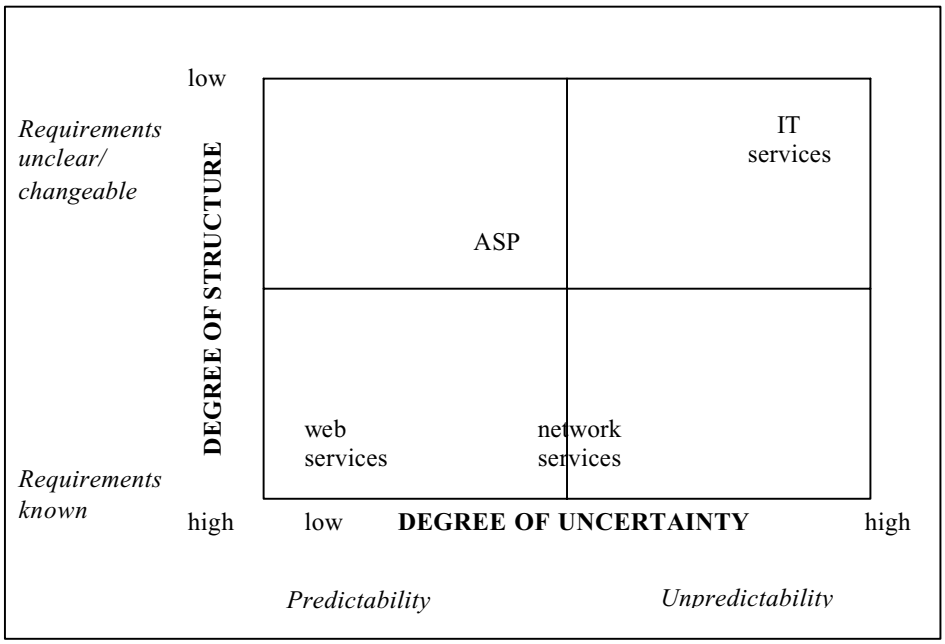


Figure 1. IT service models classified by complexity of service.

By taking a service-oriented view of IT-related activities and products then, it becomes possible both to describe them in a uniform way, as IT services, and to propose a classification of IT service models on the basis of their complexity. In this way we might

have a single taxonomic framework for describing, comparing, evaluating and managing IT services. Some implications of this are discussed below.

6 Some implications of an IT service framework and a taxonomy of IT services.

Clearly no single model of IT services encompasses the range of complexity of all types of IT services while providing automated management techniques. The model which provides the most extensive automated management techniques, the web services model, falls in the low complexity quadrant. Conversely, the model which does attempt to address a wide range of services (ITSM) does not attempt to automate techniques. If we look at the relationship between degree of complexity and degree of automation, it seems that there is a systematic variation in the degree of automation in relation to the degree of structure.

This classification of IT services demonstrates the need for techniques to be expanded to account for IT services of greater human, organisational, technical and environmental complexity, involving greater customisation. In this way, the benefits to be gained from such techniques can be more widely applied.

That said, one of the striking characteristics of web services is the direction of developments. The work currently being undertaken in SLA languages, referred to in earlier discussion here for example, extends the possible application of the web services model into areas of greater complexity. This trend could continue for a number of reasons, discussed below, and a contextual framework might aid our understanding of this as the web services model moves through categories in the taxonomy.

6.1 Extension of the web services model into areas of greater complexity

The possibly exists for the web services model to remain structured enough so that standards can be implemented and computational systems developed for them but become robust enough to deal with, for example, greater uncertainty over time. One way in which this might happen is that services might be broken down in to their component elements and distinctions made among those components on the basis of their complexity. Areas subject to negotiation and uncertainty may be isolated from others which could then be treated more like web services. In this way, services which are exposed to uncertain risks of failure and change, or to political and economic vagaries either directly or in their

dependencies, may be able to benefit from standardisation of at least some components of the service.

6.2 Improved understanding of complex services

As Keen and Scott Morton [22] comment, a task which appears unstructured may not be inherently so but may appear so due to lack of knowledge. It is possible then that services which currently appear to be complex may appear less so as our understanding of services increases through the filter of a service framework. Thus they may become more amenable to automation and standardisation. Increased knowledge of services may also come through techniques such as automated information gathering and analysis as a function of service management (eg runtime moderation of service levels) and of the embedding of service management data at the device level.

6.3 Increased sophistication of the web services model

As the underlying processes of service management become better understood, they tend to become refined, standardised and automated. For example, the developments in SLA management in web services represent such a process refinement.

6.4 Cross application of techniques among models

Just as SOA provides a framework for services to be decomposed and integrated flexibly, a framework for service models may assist the decomposition of elements and techniques of service models and their recombination in other forms.

A single framework enables us to view services in relation to each other; to see what techniques used in one quadrant might be developed in another. As automated processes such as SLA management become more sophisticated (capturing the semantics of a service for example), these techniques may be applied to more complex services. For example an ASP service such as Salesforce.com might eventually become a web service. A further examination of such issues may well make for a richer taxonomy with well defined boundaries.

Extensions of techniques and their benefits from one model to another might also include, for example, the expansion of the capability of the UDDI to encompass business rules to increase the sophistication of negotiation relating to web services and allow for the kind of customised conditions of service of the ITSM model. The lifecycle support processes of the ITSM model might also be addressed by the web services model, as might the resource mapping of the network services model. In the contracting of services,

or hierarchies of services, the implementation of automated methodologies in the ITSM model could contribute to the efficiency of IT service management

6.5 Locating services within a single framework

A single framework might also enable us to make distinctions between both significant similarities and significant differences among services. This might help, for example, in the choice of appropriate strategies for developing, managing or selecting different types of services. Or, it might distinguish issues that are similar across services, regardless of the degree of complexity, such as SLAs and service management processes.

6.6 Role of SLAs

In a service view, the SLA is a central component of the service. It references all the other elements and components of the service. A cross-model view of SLAs within a service framework may yield findings and techniques of significance to service management, the measurement of compliance with SLAs and the evaluation of service levels.

6.7 Managing IT services

In managing IT services and the contracting of services, a common, integrated framework encompassing the range of IT service, enabling services to be categorised, might serve to reduce ambiguity and assumption about our understanding of commonly used terms as well as to inform the evaluation and negotiation of contracted services and assist in the process of risk assessment.

6.8 Standardisation

Standardisation of terminology, processes, technique and practice relating to IT services, as with any standardisation, will likely produce economies of effort and increase flexibility of the application of resources, both human and technical. Standardisation of models across categories could bring benefits in terms of increased agility, better economy and improved risk management in services.

6.9 Redesign of business processes

The impact of the web services model, and a single services framework on organisational design may be significant in structuring business problems in a way that makes them more amenable to a service view and web service-type applications. Just as the models embodied in customer relationship applications, enterprise resource planning and supply chain management have changed and to some degree standardised the way organisations are structured, so might these new service models influence the refinement of the structures when seen from service viewpoint. As organisations become more structured and less complex, they are able to take advantage of the kinds of standardisation and automation to be found in the models discussed here.

Conclusion

The discourse of web services has appeared to be dominated by a technical discourse at the expense of some of the complex organisation issues addressed by other models. However, when viewed in the broader context of the diversity of IT service models, the future path of web services clearly has many possibilities to offer the broader range of IT services.

References

1. The Stencil Group, I., *UDDI.org White Paper: The evolution of UDDI*. 2002.
2. Gottschalk, K., *Web Services architecture overview*. 2000, IBM.
3. UDDI.org, *UDDI Models: Classification Schemes, Taxonomies, Identifier systems and Relationships.* 2003.
4. Cerami, E., *Web services essentials*. 2002, Beijing ; Cambridge, MA: O'Reilly. xiii, 288.
5. Arsanjani, A., et al., *Web Services: Promises and Compromises.* Queue ACM Press, 2003. **1**(1): p. 48--58.
6. Dan, A., H. Ludwig, and G. Pacifici, *Web service differentation with service level agreements*. 2003, IBM developerWorks.
7. Ludwig, H., et al., *A Service Level Agreement Language for Dynamic Electronic Services.* Electronic Commerce Research, 2003. **3**: p. 43-59.
8. Lamanna, D.D., J. Skenne, and W. Emmerich. *SLAng: A Language for Defining Service level Agreements*. in *Proceedings of the IEEE Computer Society Workshop on Future Trends of Distributed Computing Systems*. 2003: IEEE.

9. Williamson, O.E., *Markets and hierarchies, analysis and antitrust implications : a study in the economics of internal organization*. 1975, New York: Free Press. xvii, 286.
10. Park, J.T., J.W. Baek, and J.W.K. Honk, *Management of service level agreements for multimedia Internet service using a utility model.* IEEE Communications Magazine, 2001. **39**(5): p. 100-106.
11. Koch, C., *Put IT in Writing: Service Level Agreements*, in *CIO Magazine*. 1998.
12. Patnayakuni, R. and N. Seth. *Why License When you Can Rent? Risks and Rewards of the Application Service Provider Model*. in *Special Interest Group on Computer Personnel Research Annual Conference*. 2001. San Diego CA USA: ACM.
13. Greschler, D. and T. Mangan, *Networking lessons in delivering 'Software as a Service'- Part 1.* International Journal of Network Management, 2002. **12**: p. 317-321.
14. CCTA, *An introduction to ITIL and the IT Infrastructure Library*. 2001, London: Her Majesty's Stationary Office Books.
15. Lewis, L. and P. Ray. *Service level management definition, architecture, and research challenges*. in *Global Telecommunications Conference, 1999. GLOBECOM '99*. 1999: General or Review.
16. van Bon, J., *The Guide to IT Service Management*. Vol. 1. 2002, London: Addison -Wesley.
17. Dekker, J. and L. Hendriks, *Best practice in acquisition and procurement management: the Information Services Procurement Library*, in *The guide to IT service management*, J. van Bon, Editor. 2002, Addison -Wesley: London. p. 277-296.
18. Bhoj, P., S. Singhal, and S. Chutani, *SLA management in federated environments.* Computer Networks-the International Journal of Computer and Telecommunications Networking, 2001. **35**(1): p. 5-24.
19. Rodosek, G.D. *Quality Aspects in IT Service Management*. in *DSOM 2002 LNCS 2506*. 2002: Springer-Verlag.
20. Garschhammer, M., et al. *A case-driven methodology for applying the MNM service model*. in *Network Operations and Management Symposium, 2002. IEEE/IFIP*. 2002.
21. Lewis, L. and P. Ray, *On the migration from enterprise management to integrated service level management*, in *2001 Enterprise Networking, Applications and Services Conference Proceedings - Entnet(at)Supercomm2001*. 2001. p. 17-24.
22. Keen, P., G. and M.S. Scott Morton, *Decision support systems : an organizational perspective*. Addison-Wesley series on decision support. 1978, Reading, Mass.: Addison-Wesley Pub. Co. xv, 264.

.

Extending Web Service Technology towards an Earth Observation Integration Framework

Marcello Mariucci[1,2,*] and Bernhard Mitschang[2]

[1] European Space Research Institute, European Space Agency,
00040 Frascati (RM), Italy
[2] Institute of Parallel and Distributed Systems, University of Stuttgart,
70569 Stuttgart, Germany
{mariucci, mitsch}@informatik.uni-stuttgart.de

Abstract. In this paper we describe the implementation of a service-based application integration solution for the complex domain of Earth Observation (EO) application systems. The presented approach is based on an EO integration framework. It supports the concatenation of disparate software applications to flexible EO process chains. Resulting EO services are provided to end users by means of Web Service technology. We demonstrate that current standard technology is not sufficient to dynamically publish and interactively invoke EO services over the Web. We describe necessary extensions and adaptations.

1 Introduction

An Earth Observation (EO) integration framework is a service-based application integration approach for the interdisciplinary domain of EO application systems [1]. It facilitates the development and execution of EO services by supporting the integration of disparate applications to EO process chains. EO services are based on the intensive use of large data sets from space. They process raw EO data sets into increasingly specialized products until a certain level of quality is achieved. This processing requires the tight cooperation of several distributed experts, and intensive computation across a coordinated sequence of both, interactive and automatic processing steps. Appropriate examples for such EO services are the generation of weather forecasts, forest fire detection, and oil slick monitoring. Fig. 1 illustrates a simplified process flow of a representative EO service. It depicts the generation and use of forest fire observation products.

Radar images and meteorological vectors are regularly sensed and acquired by related acquisition stations ①. *Data Providers* monitor the creation of these EO data sets, and ensure their correct ingestion into the Central Server ②. The Central Server pre-processes the received data for fire detection purposes, and forwards them to an

[1] Work was done while the author was visiting the European Space Agency (ESA/ESRIN).

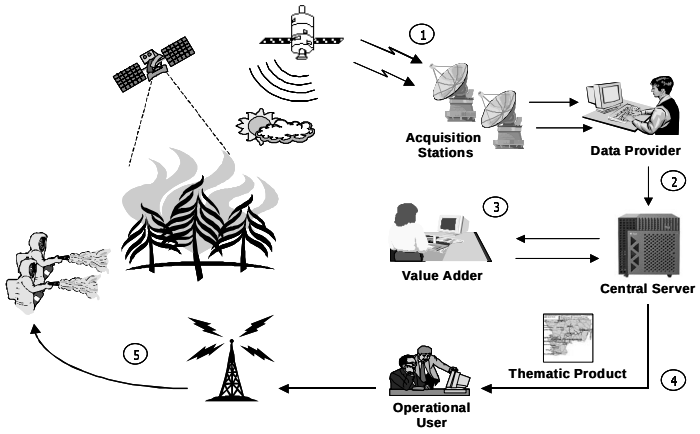


Fig. 1. Oversimplified EO Service for Forest Fire Detection

appropriate EO imagery expert. This so-called *Value Adder* analyzes, filters, maps, and merges the processed data, and eventually extracts fire observation parameters like burned area contour and perimeter ③. Analysis results are then sent back to the Central Server where they are packaged into thematic products. Generated thematic products are sent to *Operational Users* ④, who assess and validate them by using locally available information ⑤. Based on these results, they can initiate and coordinate operative actions, such as fire-fighting and rescue operations ⑥.

The example scenario shows that the seamless processing of EO services requires the integration of heterogeneous tools and management systems. That is, transitions between processing steps have to be automated and coordinated. Furthermore, data set representations have to be based on a common model. The European Space Agency has approached with mainly two research directions the implementation of such an integration solution. In the following Section 2 we briefly outline and analyze these initiatives. In Section 3 we present our solution of an EO integration framework. We introduce the service-based application integration approach, and highlight the related integration model. In Section 4 we describe the realization of our integration framework. We emphasize the use of Web service technology, and present necessary extensions and adaptations to dynamically publish and interactively invoke EO services over the Web. Section 5 summarizes and concludes the paper with an outlook to further work.

2 Related Work

The European Space Agency (ESA) has approached with mainly two activity directions the development of an EO integration framework. On the one side it has pursued a *top-down approach*, which has analyzed the extension of an existing EO application infrastructure towards a generic, multi-application platform. On the other side, it has investigated a *bottom-up approach*, which has developed an interoperable

protocol environment for coupling interdisciplinary EO facilities to integrated solutions.

ESA's top-down initiative is introduced and analyzed in [1]. The basic idea of this approach is to reuse generic functions of an already implemented EO application systems, and to build up a common framework for EO application systems. The architecture is based on a modular, object-oriented CORBA design, and deploys function modules on geographically distributed computing machines. EO services are created by combining appropriate modules to thematic process chains. Missing modules are added to the framework on demand. Corresponding process descriptions are hard-coded within so-called server modules, and provided as EO services via CORBA protocols. Analyses of this approach turned out that the infrastructure is extremely inflexible and not particularly suitable to form an EO integration framework. The creation of EO services represents a huge overhead, leading to twisted and hardly manageable software systems. Furthermore, appropriate process control and metadata management facilities are missing [1].

ESA's bottom-up approach is related to the development of a Multiple Application Support Service (MASS) protocol environment [2]. It defines an interoperable infrastructure that aims at coupling clients, tools, and management systems. Basically, MASS protocols extend the functionality of CORBA protocols and services for the purpose of EO application systems. Analyses concerning the suitability of this approach to form an EO integration framework revealed performance and flexibility problems. Furthermore, they emphasized that the MASS infrastructure is lacking of an information base and development environment for managing system resources and structures. The infrastructure does not provide a comprehensive overview of software elements available in the system. MASS information bases are restricted to interface specifications. Further information like architectural design of the system or deployment information of software elements are either hard-coded in the single software products or written in some documents. This makes it difficult for EO service developers to choose the right components during EO service creation and extension.

Based on experiences gained in assessing ESA's approaches, we designed and implemented an alternative solution for an EO integration framework. The following section briefly introduces our service-based integration approach.

3 Service-based Application Integration Approach

Our EO integration framework provides an open software infrastructure for supporting the inherent complexity of EO application systems. It facilitates the coordinated integration of disparate tools and management systems along EO process chains. In addition, it exposes related EO services over the Web by applying Web service technology. As depicted in Fig. 2, EO services are published and registered in a UDDI service directory ①. End users can discover EO services they are interested in by enquiring the UDDI ②. The information they retrieve from the directory suffices to localize and use the EO service ③. For each EO service a related process flow de-

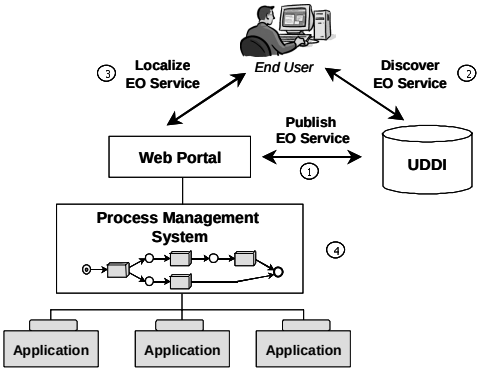


Fig. 2. Service-based EO Integration Framework

scription is available. It is used by a process management system to execute the request EO service, and to coordinate related processing steps ④.

Our service-based application integration approach is divided into three phases, i.e. design time, development time, and run time (see Fig. 3). The core element of the integration process is a shared integration model. It structures metadata about any resource, activity, and process of the framework, and represents the source from which documentation and code generation can be directly derived. The model is based on UML, and intended for process-oriented application integration purposes within EO integration frameworks. The main value of the model-driven EO integration framework is its flexibility and technological independency. By truly decoupling software specifications from implementation details, heterogeneous realizations of singe software products are abstracted. Thus, software products are integrated into flexible EO process chains, which can be dynamically reconfigured and customized as the environment changes.

At design time the focus is on the creation of the integration model. It is designed by integration architects once at the beginning of the integration process. The model specifies the structure of integration artifacts for the persistent storage of related instantiations. Development begins after the integration model is made available. Integration developers use client tools to collaboratively instantiate the integration model. Applications are thus registered, linked to their respective implementations, and assembled to combined EO services. The integration model is then used to generate appropriate code for EO service publication and execution. At run time, end users employ browser tools and proprietary clients to discover, locate, and invoke EO services. Requests are forwarded to the execution engine, which orchestrates the corresponding EO service processing.

4 Prototype Implementation

Based on the presented service-based integration approach, we implemented an EO integration framework as part of the ARSENAL project [3]. The project aimed at the

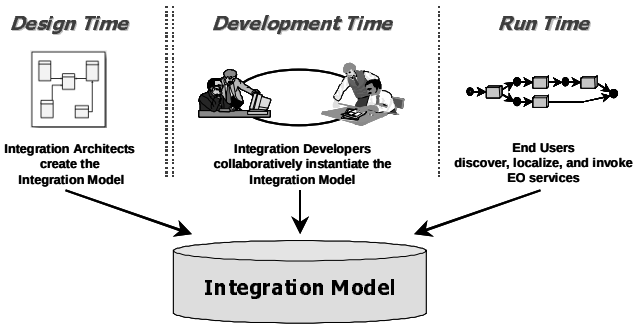


Fig. 3. Application Integration Process

design and implementation of a framework infrastructure for the flexible integration of applications within the EO domain. The main requirements were the dynamic reuse of application functions, the life cycle management of EO service structures, and the ubiquitous access to and controlled execution of EO service instances. Furthermore, the infrastructure had to be built upon commercial products, in order to guarantee best possible system stability, reliability, and low development costs. In this section, we evaluate middleware technologies regarding their use within our framework prototype. Then, we describe each integration phase in detail, and evaluate commercial middleware technologies regarding their use within our framework prototype. We emphasize selected implementation issues required for the seamless combination of those middleware products.

4.1 Middleware Technology Analysis

Metadata created during the integration process needs to be consistently stored and managed during the entire software life cycle. Its right treatment seriously enhances the integration process in terms of dynamics, flexibility, and adaptability to the constantly changing business environment. *Repository* is a technology that deals with the whole spectrum of metadata management. It provides a centralized, persistent storage, helps in reducing redundancies and inconsistencies, and improves reuse. Besides basic database management functions, the repository provides functions for the seamless management of metadata throughout the entire software life cycle (e.g. versioning, configurations). Furthermore, it offers indispensable functions for the collaborative management of system metadata (e.g. workspaces, contexts). The main value of the repository technology, and its functions are summarized in [4].

EO services describe process flows between applications within the EO domain. Different forms of middleware have been introduced by the computing community to enable integration and automation of processes. In general, a *Process Management System (PMS)* provides a central point of control for defining process flows and orchestrating their execution. It records the execution state of the process, and routes requests to applications to execute tasks [5]. In order to unify the access to applications and EO services standard interface definitions are required. Finally, *Web Ser-*

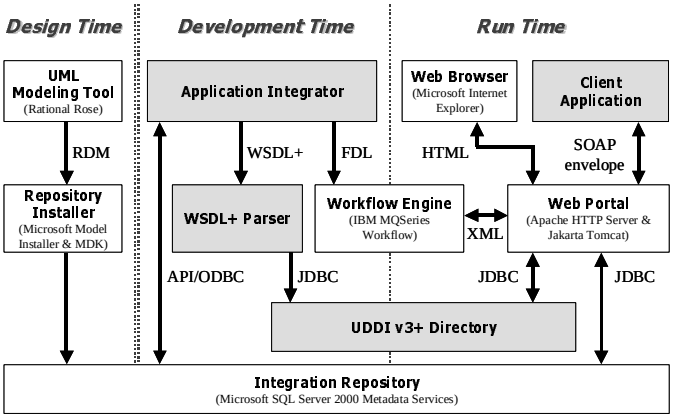


Fig. 4. ARSENAL Implementation Architecture

vices define techniques for describing, discovering, and locating software components on a global network. These techniques are programming language- and protocol-neutral, and provide ubiquitous and transparent access to system resources [6].

The introduced middleware technologies basically meet the requirements on our EO integration framework infrastructure. In the following sections we discuss the use of related commercial technology products in the EO integration framework. We emphasize additional 'glue' components required for coupling these tools to seamlessly support EO application systems. Fig. 4 illustrates the ARSENAL implementation architecture. White boxes identify Commercial Off-The-Shelf (COTS) tools, whereas grey boxes identify 'glue' components. The figure represents the big picture for the remainder of this chapter.

4.2 Design Time Support

Our EO framework prototype is based on the object-oriented repository of Microsoft's SQL Server 2000 Meta Data Services (see Fig. 4). The repository provides a powerful API for the information model definition and management during the entire software life cycle. In addition, it supplies a Model Installer, and a Model Development Kit (MDK) for the Rational Rose Modeling Tool. Both tools facilitate the installation of UML models into the repository. By means of these features, integration architects initially define the integration model by using UML, export it as a Repository Distributable Model (RDM) file, and install it in the integration repository. This procedure is done once at the beginning of the integration process.

4.3 Development Time Support

At development time, integration developers are supported to collaboratively instantiate the integration model. The Application Integrator tool (see Fig. 4) supports

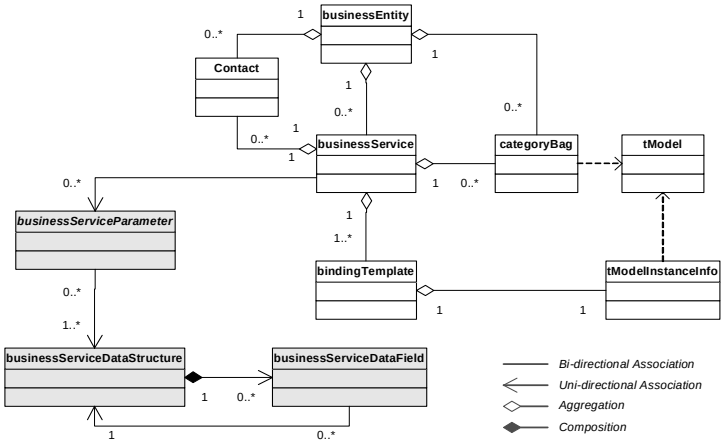


Fig. 5. UDDI v3+ Data Model

integration developers to collaboratively create EO services. It provides a user-friendly GUI, and uses the repository API to create, navigate, and manipulate the integration model content, basically by inserting, deleting, and updating object instances. In addition, it enables asynchronous collaborative work by employing Microsoft's repository features, such as version, configuration, and workspace management. The Application Integrator tool also supports the generation of appropriate, technology-specific code for publishing and executing EO services. For EO service execution purposes, it creates a Flow Definition Language (FDL) file, and exports it to the workflow engine. For EO service publishing purposes, it generates an extended Web Service Definition Language (WSDL+) file. This WSDL+ file is then parsed and imported into the UDDI v3+ registry. Extensions regarding the WSDL and the UDDI specifications are required for the interactive Web service invocation, and for the automatic registration of EO services into the UDDI registry. They are described in detail in the following subsections.

UDDI Extensions for Interactive Web Service Invocation. EO services interfaces are described as Web services and published in a private UDDI registry. This UDDI registry is a database infrastructure that enables end users and their client applications to quickly and easily find EO services [7]. It links each EO service description to its corresponding WSDL interface definition file, which abstractly describes related access structures. By means of this file, end users can integrate the EO service within their application code, and locate and invoke it through remote procedure calls.

Besides this standard form of invoking EO services, the ARSENAL prototype facilitates the interactive invocation of EO services from within a Web browser tool. That is, end users are not forced to retrieve the corresponding WSDL interface definition file to access the EO service, but are enabled to directly and interactively invoke the EO service from within their standard Web browsers.

Two approaches were analyzed to implement this feature within the ARSENAL prototype. The first approach was related to the enhancement of the Web portal to process WSDL interface definition files. As soon as an end user requests an EO service invocation, the Web portal parses the corresponding WSDL file, and dynamically generates a Web page requesting the EO service input parameters. It then sends the request together with its parameters to the corresponding port to automatically launch the EO service execution. However, this solution resulted to be too time-consuming, since the WSDL file is processed and parsed at run time. Consequently, the second approach regarded the extension of the UDDI registry to additionally manage WSDL interface definitions. That is, the UDDI registry is enhanced to enable the storage and retrieval of EO service access structures. In this way, WSDL files can be parsed at development time, and EO service interface definitions directly used to dynamically generate a Web page requesting the EO service input parameters. Although this approach extends the UDDI standard in a proprietary fashion, the performance and response time of the Web portal is enormously enhanced.

The ARSENAL prototype implements the second approach, and, therefore, extends the UDDI registry with EO service access structures. Fig. 5 illustrates the enhanced UDDI data model. It refers to the UDDI v3 standard and extends it by three entities, namely:

- **businessServiceParameter** for the specification of the EO service parameters,
- **businessServiceDataStructure** for the specification of EO service parameter data types, and
- **businessServiceDataField** for the specification of complex data types.

These three additional entities are highlighted in Fig. 5. The resulting UDDI registry is called UDDI v3+ registry.

Automatic Registration of EO Services into UDDI v3+. Our prototype implementation provides the feature to automatically generate both, WSDL interface and WSDL implementation files for EO services. Based on EO service descriptions stored in the repository, appropriate WSDL files are generated, parsed, and registered into the UDDI v3+ registry. Usually, the registration of Web services in a UDDI registry is performed in interactive mode, i.e. the service provider connects to the UDDI registry via a Web browser, registers its business entity, and describes all provided Web services. Since our repository includes all required information, we additionally support automatic registration of EO services in the UDDI v3+ registry. For this purpose, we had to appropriately extend the WSDL implementation file by 'business entity' and 'category bag' specifications. Fig. 6 lists a fragment of an exemplary WSDL+ implementation file. Extensions are highlighted.

The WSDL+ implementation file is sent to the WSDL+ parser. The WSDL+ parser is based on Apache's Xerces DOM parser, and verifies the syntactical correctness of the WSDL+ file. In addition, it inserts related entries into the UDDI v3+ registry.

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions ...>
  <documentation> ... </documentation>
  <import ... />
  <businessEntity>
    <name>Space Data Technology, Inc.</name>
    <description>Satellite Data Provider</description>
    <contacts>
      <contact useType="Organization Manager">
        <personName>Marcello Mariucci</personName>
        <details>mariucci@spacedatatech.com</details>
      </contact>
    </contacts>
    <categoryBag>
      <keyedReference tModelKey="UUDI:ClAD242-9343-2321" keyName="Earth Science" keyValue="Data"/>
      <keyedReference tModelKey="UUDI:D4EG343-3534-2316" keyName="Data" keyValue="Provider"/>
    </categoryBag>
  </businessEntity>
  <service ...>
    <documentation> ... </documentation>
    <categoryBag>
      <keyedReference tModelKey="UUDI:D3ER342-2345-6545" keyName="Data" keyValue="Ingestion"/>
    </categoryBag>
    <port ... > ... </port>
  </service>
</definitions>
```

Fig. 6. WSDL+ Implementation File

Generation of a Business Process Definition File. The Application Integrator tool is capable of exporting workflow specifications of EO services to a FDL file. This file includes instructions for the workflow engine to properly execute EO services at run time. To prepare EO service executions, such a file is generated and imported into the respective BPMS. FDL is a proprietary workflow specification format of IBM. However, the Application Integrator tool is open and flexible enough to adapt its code generation procedure to other workflow description formats, such as the emerging BPEL [8] format.

4.4 Run Time Support

At run time, end users discover, locate, and invoke EO services by applying Web service technology. By means of a Web browser, end users query the UDDI v3+ registry to retrieve information about available EO services and their access locations. More specifically, the Web browser connects to the Web portal and requests an appropriate Java Server Page (JSP). The request is forwarded to the Servlet engine which executes corresponding processing steps, queries the UDDI v3+ registry, and sends related information back to the end user via the Web portal. The UDDI v3+ registry provides descriptive information about EO service interfaces and access points. End users use this data to locate EO services, and to invoke them by specifying related parameters.

EO services are offered through the SOAP (Simple Object Access Protocol) [9]. SOAP messages for the invocation of EO services are either created by client applications, or automatically generated by the Web portal for interactive EO service invocations. They are then sent to the corresponding access point by using the HTTP post protocol. Through this access point the message is forwarded to the SOAP adapter, which processes it, and sends corresponding execution instructions to the

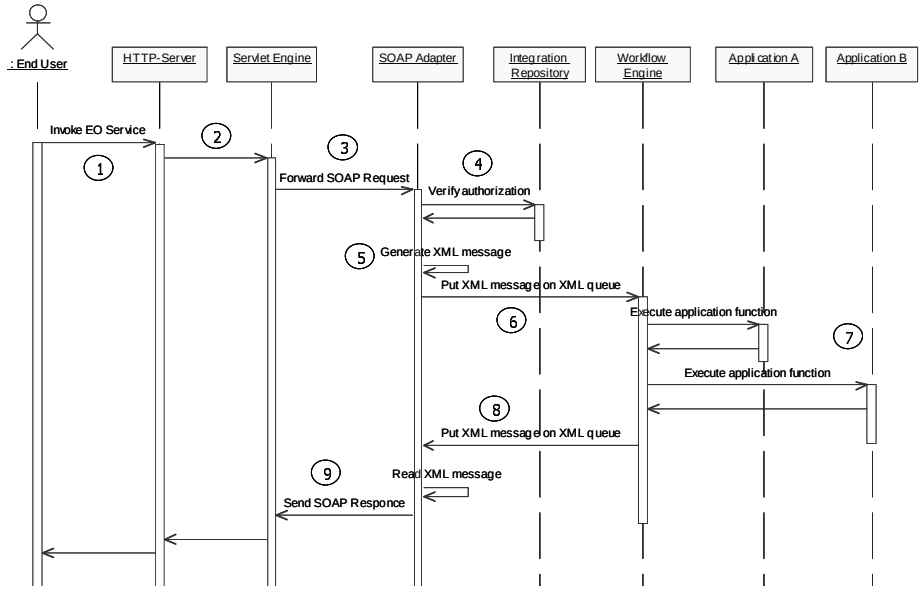


Fig. 7. UML Sequence Diagram for EO Service Invocation Handling

workflow engine. Execution instructions are coded in the Extensible Markup Language (XML) format [10]. The workflow engine provides a reliable and robust infrastructure for the orchestration of EO services related process chain. Results are sent back to the Web portal, which either forwards them to the client application, or sends them via email to the end user. In the reminder of this section, we briefly describe the dynamic invocation of EO services via SOAP messages.

Dynamic SOAP Adapter for a Workflow Management System. As described above, end users can send SOAP request messages to an access point in order to initialize EO service executions. For that purpose, the Web portal integrates a dynamic SOAP adapter, which is able to parse SOAP messages and to forward related execution instructions to the workflow engine. Our SOAP adapter is based on Apache SOAP, which per se provides a SOAP handler to parse SOAP messages and to map requests to static Java classes. Unfortunately, this implementation approach does not meet our demands, as in our integration infrastructure Web services are dynamically created, and, thus, Web service related Java classes do not exist in advance. A new dynamic SOAP handler, which additionally adapts to a workflow management system, was implemented. Fig. 7 depicts a UML sequence diagram regarding the invocation of EO services via SOAP in our EO integration framework.

The end user sends a SOAP message to the HTTP Server ①, which forwards the requests via Servlet engine ② to the SOAP adapter ③. The SOAP adapter verifies the request. In order to check whether the service parameters are correct and whether the client is authorized to execute the EO service, the adapter interacts with the integration repository ④. This connection is also used to get service execution instructions. The SOAP adapter generates XML statements about these execution instruc-

tions along with the service parameters ⑤, and sends them via XML queues to the workflow engine ⑥. The workflow engine starts the EO service among the involved applications ⑦. At the end, it sends a status report back to the SOAP adapter ⑧, which forwards it to the end user via SOAP ⑨.

5 Conclusions and Future Work

In this paper we presented a flexible EO integration framework for supporting the inherent complexity of EO application systems. We focused on a service-based application integration approach and its realization. We emphasized our prototype implementation, which integrates and adapts COTS tools for workflow management, repository, and Web services technology. The prototype infrastructure has been successfully applied in real EO environments, which proves that flexible application integration could be achieved by the right composition and adaptation of existing technology. Up to now, the EO integration framework infrastructure is limited on a few scientific applications and a selective user community. However, visions concern the use of such frameworks to foster commercial EO applications, and to build up a market for EO data and application providers.

In our approach, Web service technology is not used to link applications inside the EO integration framework. Since Web service technology products are still in an experimental fashion, we preferred to use more expensive, but proven and reliable, products. Future work regards the employment of Web service technology for integrating internal key systems. Further future work is concerned to emerging Grid technologies [11]. As of now, Grid technology is perceived as an orthogonal issue. Coupling Grid with our EO integration framework offers the possibility of transparently executing application functions on best possible resource allocation configurations. Furthermore, security solutions of local applications could be maintained, and intensive computation applications executed on dynamic high performance resources. The employment of Grid would greatly simplify the sharing and dissemination of applications, and enhance the quality, effectiveness and efficiency of our application integration approach.

References

1. Mariucci, M., Mitschang, B.: On Making RAMSES an Earth Observation Application Framework. In: Smari, W.W., Melab, N., Chen, S.-C. (eds.): The 2nd International Conference on Information Systems and Engineering. The Society for Modeling and Simulation International, Vol. 34, Nr. 2. San Diego (2002) 67–72.
2. Doherty, C., Usländer, T., Landgraf. G.: Multiple Application Support Services. An ESA Protocol for EO Application Clients. In: Earth Observation & Geo-Spatial Web and Internet Workshop. Committee on Earth Observation Satellites. London (2000). http://webtech.jrc.it.

3. European Space Agency, University of Stuttgart: ARSENAL Project. Official Homepage. http://www.informatik.uni-stuttgart.de/ipvs/as/projekte/arsenal/index_engl.html.
4. Bernstein, P.A., Dayal, U.: An Overview of Repository Technology. In: Bocca, J.B., Jarke, M., Zaniolo, C. (eds.): 20th International Conference on Very Large Data Bases. Santiago (1994) 705-713.
5. Dayal, U., Hsu, M., Ladin, R.: Business Process Coordination: State of the Art, Trends, and Open Issues. In: Apers, P.M.G., Atzeni, P., Ceri, S., Paraboschi, S., Ramamohanarao, K., Snodgrass, R. (eds.): 27th International Conference on Very Large Data Bases. Rome (2001) 3-13.
6. Leymann, F.: Web Services: Distributed Applications without Limits – An Outline. In: Weikum, G., Schöning, H., Rahm, E. (eds.): 10th BTW 2003 Datenbanksysteme für Business, Technologie und Web. Leipzig (2003) 2-23.
7. Belwood, T., et al.: UDDI 3.0. UDDI Spec Technical Committee Specification. http://www.uddi.org.
8. Leymann, F., Roller, D.: Business processes in a Web service world. A quick overview of BPEL4WS. IBM report. http://www-106.ibm.com.
9. World Wide Web consortium (W3C): Simple Object Access Protocol (SOAP) 1.1. W3C Note 08 May 2000. http://www.w3.org/TR/SOAP
10. World Wide Web consortium (W3C): Extensible Markup Language (XML). http://www.w3c.org/XML/.
11. Foster, I., Kesselman, C.: The Grid: Blueprint for a New Computing Infrastructure. Morgan Kaufmann (1999).

DoJa: Service oriented application platform for mobile phones using Java

Masayuki Tsuda[†], Atsuki Tomioka[‡],
Takefumi Naganuma[‡] and Shoji Kurakake[‡]

[†]Customer Equipment Development Department, NTT DoCoMo, Inc.
[‡]Multimedia Laboratories, NTT DoCoMo, Inc.
3-9-11 Hikari-no-oka, Yokosuka, Kanagawa, Japan
tsuda@cet.yrp.nttdocomo.co.jp
{tomioka, naganuma, kurakake}@mml.yrp.nttdocomo.co.jp

Abstract. DoJa is a major application platform that is designed for mobile e-services that will run on DoCoMo's mobile phones and its wireless network. It ensures that our users can easily receive advanced services on their palm. Since DoJa services started in 2001, the number of users has exceeded 17 million, service-oriented functions have been added to DoJa, and the number of DoJa applications now exceeds 7000. This paper mainly describes the two key features of DoJa that are specific to mobile phones and wireless networks. One is a framework for applications that will be held on a mobile phone and that behave like an agent. The other is a new object that protects the user's privacy. We elucidate the problems and mechanism needed to realize these features in a mobile phone. Their usefulness is confirmed by experiments.

1 Introduction

The mobile phone is changing its role from voice-only communication to mobile e-services. In Japan, users began accessing the Internet via mobile phones in 1999. Since that time, mobile e-services such as banking, ordering books and reserving tickets have been flourishing and are now major components of e-business. In Europe, similar mobile e-services were started in 2002 and the number of the users has increased rapidly. This movement is spreading over the rest of the world.

One of the key technologies for realizing mobile e-services is Java[1][2]. A small Java program on a mobile phone can, when executed, collaborate with applications in distributed servers over a wireless network and the Internet. Services are dynamically composed to realize various mobile e-services in the user's palm. Java for mobile phones is a minimum subset of Java for personal computers, Java 2 Platform Standard Edition, and is designed for kilo-byte environments. That is, application size, memory size and so on are of the order of kilo-bytes because the CPU power and memory of mobile phones are much less than those of personal computers or workstations[3]. Java for a mobile phone is termed Java 2 Platform

Fig. 1. These are the latest DoJa enabled mobile phones. The applications realized range from mobile e-services to ubiquitous services.

Micro Edition (J2ME) and consists of a configuration and a profile. The configuration is called Connected, Limited Device Configuration (CLDC)[4], which consists of a virtual machine (VM) and a minimum subset of basic Java class libraries. The profile consists of device-oriented Java class libraries such as user interface, service-oriented Java class libraries for mobile e-business, and an application management function. Accordingly, to realize attractive mobile e-services using J2ME, it is necessary to develop profiles, especially the service-oriented Java class libraries, from a service-oriented viewpoint.

DoJa[5] has a service-oriented profile from which many successful mobile e-services have been developed. It was designed based on analyses of concrete business requirements and case studies. Its design also allows for the full utilization of the features of our mobile phones and our wireless network. Since the first release of DoJa in 2001, the number of its applications such as e-commerce, games, information delivery, and karaoke has reached 7000, and the number of users now exceeds 17 million. As a result, DoJa has become a major application platform in our mobile e-services(Figure 1).

This paper describes the two key functions of DoJa: a framework for stand-by applications and XObject. A stand-by application, we call them "machiuke" applications, is an application that runs constantly like an agent or a Unix daemon. To realize this kind of application, two problems have to be solved. One is power consumption. The other is switching key inputs to the appropriate application or phone function. The former is serious because Java consumes battery power very quickly. The worst mobile phone's power consumption leaps about 20 times when a Java program is run constantly. Thus some power saving mechanism is needed to run Java constantly. The latter is closely associated with usability. When Java doesn't run, users can make a phone call or email by pressing keys smoothly. But, when Java runs, all key inputs are delivered to Java and they must first terminate it to make a phone call or send an email. This degrades usability. To solve these problems we introduce three states to the

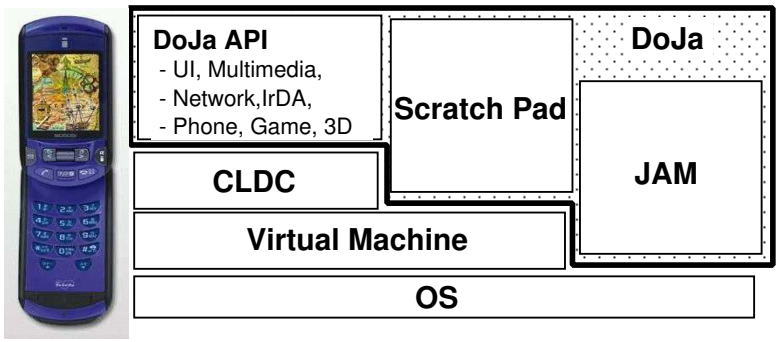


Fig. 2. A structure of DoJa

application life-cycle. Making the application switch among these three states yields power savings and key input switching. XObject was developed to protect the user's private data from leaking from the mobile phone. If the user's private data in an address book is treated as an ordinary Java object, it is easily leaked by sending the object to a server. Thus, it is necessary to implement some mechanism that prevents object release. Our solution is the new Java object called XObject.

In the following sections, we first overview DoJa. We then detail the framework of machiuke applications and XObjet. Experiments conducted to evaluate the usefulness of our proposal are then described. Finally we have provide some concluding remarks and discuss feature work.

2 An overview of DoJa

2.1 Composition of DoJa

As shown in Figure 2, DoJa consists of three main parts. Most J2ME related technologies have been developed as part of Java standardization activities, but all components in Figure 2 except for CLDC and VM were developed by DoCoMo to realize our mobile e-services. DoJa components are as follows.

- **Java Application Manager (JAM)**
  JAM manages downloading, updating, removing, invoking and terminating applications. Because there is no Windows command prompt nor Unix shell in a mobile phone, JAM actually provides the equivalent functionality. Only one application can be invoked in DoJa at the same time because of the memory limits; this restriction is also managed by JAM. Moreover, it manages security during application runtime. For example, DoJa prevents the application from communicating with any other server than the one it was downloaded from. This prevents the all-too frequent DoS attack. To do this, JAM memorizes the server's URL from which it was downloaded and always

compares the URL being accessed against the memorized URL. If the URLs are different, JAM asks the VM to issue a SecurityException to prevent the communication.

- **DoJa APIs**
  DoJa APIs consist of a user interface, network functions such as HTTP/HTTPS and IrDA Obex, multimedia functions, phone specific functions and other application-oriented functions like 3D graphics and gaming. We have provided the APIs with application life-cycle and API behaviors. These APIs were developed by analyzing actual service requests and concrete case studies provided by our business team. Service developers actually develop their DoJa application using these APIs. In developing these APIs, we took several points into consideration. They included signatures of DoJa APIs and the user interface. Regarding signatures, we made them similar to those of existing Java API equivalents. This makes it easy for the developers, who are familiar with Java, to discern the functionalities or behaviors of the DoJa APIs. Regarding the user interface, it was designed to be broadly mimic that of the browser implemented prior to DoJa. Since the users were familiar with this interface they experience no confusion. When an application is invoked, it is dynamically linked to these APIs and CLDC in the mobile phone and is executed on a VM.

- **Scratch Pad**
  Scratch Pad is a data storage object implemented in the internal memory of the mobile phone, one pad is assigned to each application. The application can access only its assigned Scratch Pad. This restriction is managed by JAM; JAM memorizes the pairs of applications and Scratch Pads. When an application is invoked, JAM cleans the memory, loads only the application's Scratch Pad into memory and gives access rights to the application. This access control allows copyrighted material like game characters or ringing melodies to be safely stored in phone; other applications can't access or use them.

By combining these three parts with CLDC and VM in a mobile phone, DoJa applications can be executed with safety. The key advances in DoJa are JAM and the DoJa APIs. These two technologies allow new functions specific to mobile phones or mobile e-services to be realized easily.

2.2 Sequence of downloading DoJa application

DoJa applications can be downloaded from servers of application providers in the Internet over the air. That is, a user of a DoJa enabling mobile phone can download and execute the applications at any time and anywhere. A server from which the applications are downloaded is not limited to DoCoMo's sites only and everybody can develop and provide DoJa applications. The applications can be found using a compact HTML browser.

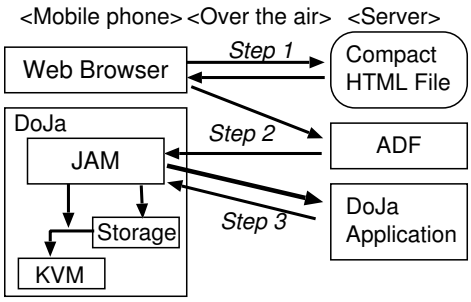


Fig. 3. Download sequence of DoJa application

Figure 3 shows the sequence of downloading DoJa applications from a server in the Internet to a a DoJa enabling mobile phone. The download squence are the following three steps.

Step 1. A user download a compact HTML file to the mobile phone and accesses the URL written in that file using the compact HTML browser.

Step 2. By accessing the URL, an application descriptor file (ADF) is downloaded and is automatically handed to the JAM. The ADF contains the information of the DoJa applicationl, that the user want to download, such as the applcation size, the scratchpad size, the version of the application and the main class. Before downloading of the DoJa application, JAM checks the ADF and investigates whether the DoJa application can be executed in the mobile phone or not.

Step 3. When the DoJa application turns out to be able to run in the mobile phone, the JAM automatically downloads and executes it.

Downloading the application over the air is charged. If the downloaded DoJa application does not run well, the unnecessary money is charged for the user. To prevent this matter, the JAM checks the ADF in the Step 2 before downloading the DoJa application.

The next section provides details of how JAM and APIs can realize a machiuke application and XObject.

3 Service oriented functions in DoJa

3.1 A "Machiuke" application

3.1.1 What is a machiuke application?

"Machiuke"[1], refers to the state in which a user can make a call or catch a call at any time. After a mobile phone is turned on, most of the time it is in

[1] Standby is the common translation of machiuke. For example, standby is used in English catalogs like "the standby time of this phone is 180 hours".

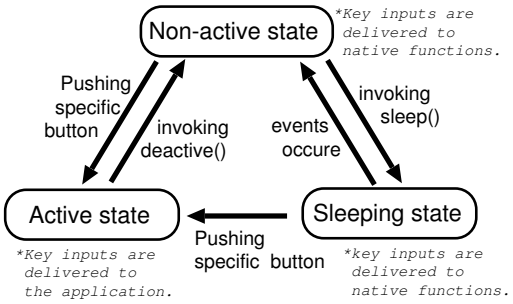


Fig. 4. New states and transitions for a machiuke application

the machiuke state. To utilize the machiuke state for mobile e-services, we have developed a mechanism for executing machiuke applications and developed the necessary APIs. A machiuke application is a general name that refers to an application that can constantly run in the machiuke state. What we have developed is a framework for developing machiuke applications. The actual machiuke applications are to be created by service developers using the APIs.

A typical example of a machiuke application is an news agent. It constantly runs to automatically gather the latest information like news and stock prices by periodically accessing a server, and showing the information on the phone's display. By using the agent, users can get these latest information whenever they look at the display. If the agent targets location data, it also can provide the user with location-based information such as advertisements of surrounding shops.

3.1.2 Three states and transitions of a machiuke application

When realizing a machiuke application, the most important thing is that the user must be able to make or receive calls at any time, even while the machiuke application is active. As mentioned in Section 1, the key requirements in developing any machiuke application are minimal power consumption and switching key inputs. These problems are unique to mobile phones because they have small batteries and only a single window system. To achieve these goals, as shown in Figure 4, we have defined the following three states in an application's life-cycle and have developed a mechanism of transitions among the three states while the machiuke application is active. Details of the states are as follows.

- **Active state**
  In this state, the machiuke application runs an ordinary Java application. All key inputs are delivered to the application's VM and handled by the application.
- **Non-active state**
  In this state, all key inputs are delivered to the basic phone functions. As soon as the user pushes a key,the application's window that is replaced by

a window for the basic phone functions like phones call or email. The user can use these native phone functions without bothering to terminate the application.

- **Sleep state**
  The sleep state is tended to minimize the power consumption. In this state, not only the machiuke application itself but also its VM is suspended, so Java consumes no power at all. This sleep state realizes significant power savings. As an example, the application can be in the sleep state for 59 seconds and active for 1 second.

The transitions among these states each other are done by an application itself using APIs or event handling. The ways of doing transitions are as follows.

- **Active state ⇔ non-active state**
  machiuke applications basically run in the non-active state. The transition from the active state to the non-active state is triggered by the application's logic; the application invokes the method `deactivate()`. The transition to the active state is triggered by the user pressing a specific key, the activation key.
- **Non-Active state ⇔ sleep state**
  The transition to the sleep state is triggered by the application invoking the method `sleep()`. This is done because the practicality of power saving in an application strongly depends on the service. The transition to the non-active state is automatically triggered by one of several events. These events happen when a mobile phone is unfolded or the wakeup timer set in advance expires and so on. The reason for this is that the VM itself is suspended so only solution is to use events. These events can be received by the application using event handler method `processSystemEvent()`when the state enters the non-active state.
- **Sleep state ⇒ Active state**
  The transition to the active state is triggered when the user presses the activation key in the same way as the transition from the non-active state to the active state. The application can never transit from the active state to the sleep state. If an application could invoke `sleep()` from the active state, errors such as infinite looping between the sleep state and the active state would become possible. In the non-active state, all key inputs are delivered to the native phone functions so the user can terminate any infinite looping. In the active state, however, the application receives all key inputs and there is no way to stop it. Thus, only the transition from sleep state to the active state is permitted.

State management and these transitions are controlled by JAM. To do so, the VM notifies JAM of the invocation of `sleep()` and `deactivate()`; JAM is also notified of events such as unfolding of the mobile phone, expiration of the wakeup timer, or activation key press. According to the methods or the events, JAM suspends or resumes the thread on which the VM runs, or switches the delivery of all key inputs.

```
import  com.nttdocomo.ui.*;

public class Main extends MApplication implements SoftKeyListener {    // Main class
    Panle mainPanel;
    public void start( ) {                                              // starting this program from this method
        mainPanel = new Panel();                                        //  making a Panel object
        mainPanel.setSoftKeyListener(this);                             //  setting the SoftKeyListener in the Panel object
        Display.setCurrent(mainPanel);                                  //  showing the Panel object on the display
    }
    public void softKeyReleased(int softKey){                           //  defining the concrete method in the SoftKeyListener
        if ( softKey==Frame.SOFT_KEY_1 ) {                              //  if SOFT_KEY_1 is released,  the state is changed
            deactivate();                                               //  to the non-active state invoking deactive()
          }
    }
    pubilc void processSystemEvent(int type, int param){                //  defining the concrete method in the processSystemEevent
    if (type == MApplication.FOLD_CHANGED_EVENT){                       //  if a mobile phone is folded,  the WakeUpTimer is set and
          if (param == 0) {                                             //   the state is changed to the sleeping state invoking sleep()
              setWakeUpTimer(3000);
              sleep();
          }
    }else if (type == MApplication.WAKEUP_TIMER_EVENT){                 //  if the WAKEUP_EVENT happens,
           if (param == 0) {
               doSomethingInActiveMode();                               //  the state is automatically changed to the active mode and
                  :                                                     //  doSomethingInActiveMode() is executed
```

Fig. 5. Sample code of a machiuke application

3.1.3 API of a machiuke application

We developed the `MApplcation` class to provide the APIs needed to program machiuke applications. This class is an abstract class of Java and provides the framework of machiuke applications. Sevice developers make their own machiuke applications by inheriting the `MApplcation` class and programming concrete behaviors of the methods in the `MApplcation`.

Figure 5 shows the sample source code of a machiuke application. As shown, the machiuke application starts by invoking `start()`. To change its state, it can handle the following four events from JAM using `processSystemEvent()`:

- `MODE_CHANGED_EVENT` is generated when a state is changed and is used to manage application state,
- `WAKEUP_TIMER_EVENT` is generated when the time set by using the `setWakeupTimer(int time)` method expires,
- `CLOCK_TICK_EVENT` is generated every minute by using the `setClockTick( boolean true_or_false)` method, and
- `FOLD_CHANGED_EVENT` is generated when the mobile phone is folded or unfolded.

The sample source code provides `SoftKeyListener` which is another key event handling interface. Through it, an application receives the specific key event which is assigned in advance and also changes its state to the non-active state by invoking `deactivate()`.

When the mobile phone is unfolded, the application receives `FOLD_CHANGED_EVENT`, invokes `sleep()` and enters the sleep state. Before invoking `sleep()`, the wakeup time is set using `setWakeUptimer()`. The application is woken up upon and enters the active state receiving `WAKEUP_TIMER_EVENT` from `processSystemEvent()`.

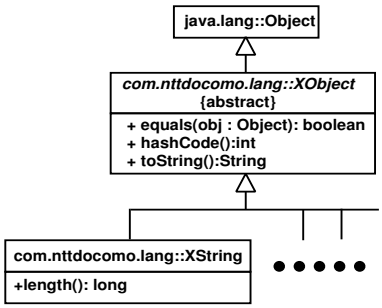


Fig. 6. Structure of XObject and its subclasses

3.2 XObject

`XObject` was developed to prevent the leakage of the user's private data via Java. The problem is that Java provides no way of preventing data leakage. For an example, when a Java application reads the data in an address book, the data are ordinarily instantiated as `String` objects in Java and these `String` objects can be sent to a server using a HTTP connection without the user being aware of the transfer. This is true for all types of objects in Java so private data is at serious risk. Thus it is necessary to develop a new type of an object that ensures that user data is handled only within the mobile phone.

To meet this requirement, we developed the main features of `XObject` as follows:(1) it can not be moved outside the mobile phone, and (2) its contents cannot be discovered by other methods. In short, the only activity permitted to an `XObject` is to display its contents on the mobile phone. What kinds of data are instantiated as `XObject` depends on DoJa's security policy and our service policy. For an example, the data of an address book in a mobile phone is user's privaicy and should be protected as an `XObject`. Although the password to access a service in a server may be a private data in one sense, it should be sent to the server and it can not be treated as an `XObject`. In DoJa, we provide the other way to protect it like SSL.

To realize these features, we have severely restricted the operation of `XObject`. The first restriction is that an `XObject` cannot be read out as byte data not can it be transferred via a stream connection in the `java.io` package. This restriction realizes the first feature. The second restriction is that an `XObject` can not be directly created using its constructor. It is only allowed to be created as the return objects of some specific methods such as the method of accessing an address book. The third restriction is that the methods used to compare the content of an `XObject` to those of other objects or casting from an `XObject` to other objects or searching `XObject` from a content always return `false`. For an example, `XObject#equals(java.lang.Object object)` checks only whether the instances are identical or not and `XObject#hashCode()` calculates the object's hash value without using the contents of the object. The fourth restriction is the restriction of the screen on which instances of `XObject` can be placed. An

XObject can be drawn on the screen in the same way as an ordinary Java object, but the methods that can acquire information from the screen on which an XObject is drawn, such as reading the pixels on the screen or saving the screen as a JPEG data, are prohibited. When these methods are invoked for the screen, a security violation is called and the application is forced to terminate. This restriction is also true for all screens on which the image object that has any XObject is drawn and for all graphics objects that include any XObject. These restrictions realize the second feature. All restrictions are implemented in XObject and related APIs. For an example, the checking methods needed to realize the fourth restriction are created with API implementation. The forced termination of an application is done by JAM. When a security violation is triggered, it is passed to JAM which forces the application to terminate.

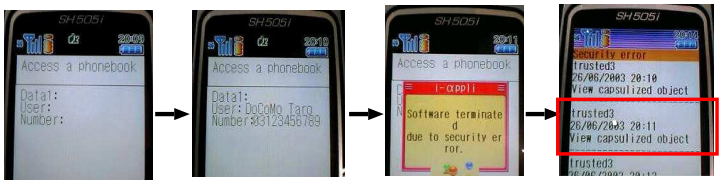
Figure 6 shows the class structure of XObject. The class of XObject consists of a base and abstract class and inheritance is set for the subclasses so the objects are defined according to data type. For example, XString is a String-type XObject that handles string data as XObject and its class is a concrete final subclass of an XObject class. In a program, a String-type XObject is directly instantiated from XString. The current version of DoJa offers only XString because only string data is treated in the actual case studies of our services. However, Image-type object for protecting digital management rights and Number-type objects such as Integer, Long, Float and Double can be realized in future extensions of DoJa in the same way as XString.

4 Experiments and evaluations

4.1 Experiments on state transition

To evaluate the states and their transitions, we conducted two experiments. One compared the power consumption of an application in the active state and that of the same application that was repeatedly switched between sleep state and non-active state. The other examined user perception of phone usability when an application in the active state was switched to the non-active state. In the experiments, we used two mobile phones: SO504i made by Sony Ericsson, Inc. and F504i made by Fujitsu, Inc.

In the first experiment, the application drew sixteen triangles at random on the display. In the active-state, it drew them every four minutes. When running as a machiuke application, it slept for four minutes, entered the non-active state, drew the triangles, and went back to sleep. The time until a full charge was completely depleted was measured and the results are shown in Table 1. In the table, the standby time means that Java didn't run and it is the time until all full power was consumed. As shown in Table 1, the machiuke application ran for about 10 or 20 times longer than the active state equivalent application. Thus, introducing the sleep state is useful in minimizing the power consumption. In the experiment, the application didn't use any network calls so the actual usage times would be shorter than those shown in Table 1. In the second experiment,



(1) The application starts.

(2) The data in the phonebook is read and shown.

(3) The application is terminated because the data was tried to be stolen.

(4) The termination is recorded in a security error log.

Fig. 7. The application is compelled to be terminated when the method is invoked to steal the contents of the XObject.

the subjects were 10 people who had not previously used a machiuke application. They were asked to make a phone call while executing an application. The application ran in the active state and in the non-active state. Eight subjects noted that the non-active state was definitely useful because they had to terminate the active state before making a phone call and they felt the step of termination was troublesome. The remaining two subjects were less bothered by the termination step. Although the number of subjects is rather small, we think that the usefulness of introducing the non-active state has been confirmed.

Table 1. Comparison of power consumption

Phone	Running in only active state	Running as a machiuke application	Standby time
SO504i	about 12 hours	about 130 hours	about 320 hours
F504i	about 6 hours	about 112 hours	about 525 hours

4.2 Experiments of XObject

We made several applications to check the XObject restrictions mentioned in Section 3, and ran them on a mobile phone. The results confirmed that all restrictions worked well. In particular, we confirmed we could not extract the XObject contents nor could we steal the contents via the screen or as graphics objects; all attempts lead to security violations which triggered application terminated as shown in Figure 7. We also confirmed that we could not compare the contents of the XObject and those of other objects. The results confirmed the usefulness of XObject.

5 Conclusions and future work

This paper has focused on the two key features of DoJa: a framework for machiuke applications and XObject. To realize a machiuke application in a mobile phone, we have solved the problems of power consumption and key input

controls by developing three states and switching between them during the application's life-cycle. The experiments described here show the usefulness of these states and the transitions. To realize a mechanism that protects the user's private data, we have developed Xobject, which prevents the extraction of data from a mobile phone; the only operation possible is to display the object's contents on the phone's screen. These two features have already been implemented in our commercial mobile phones and are now in use in actual mobile e-services.

In general, there are two directions for future mobile e-services. One is web services and the other is ubiquitous services. The main target of both is e-commerce. To realize these services in DoJa, future work on developing DoJa includes handling XML, SOAP, Bluetooth and non-contact chips.

References

1. Helal, S.: Pervasive Java, Part II. IEEE Pervasive Computing, Vol. 1, Issue 2, (2002) 85–89
2. Lawton, G.: Moving Java into Mobile Phones. IEEE Computer, Vol. 35, No. 6, (2002) 17–20
3. Riggs, R., Taivalsaari, T., and Vandenbrink, M.: Programming Wireless Devices with the Java^{TM}2 Platform (Micro Edition), Addision-Wesley, (2001)
4. CLDC:http://java.sun.com/products/cldc
5. DoJa: http://www.nttdocomo.co.jp/english/p_s/imode/index.html