Introduce: An Open Source Toolkit for Rapid Development of Strongly-Typed Grid Services

Shannon Hastings, Scott Oster, Stephen Langella, David Ervin, Tahsin Kurc, Joel Saltz
Department of Biomedical Informatics
The Ohio State University
[hastings,oster,langella,ervin,kurc,jsaltz]@bmi.osu.edu

Abstract

Service-oriented architectures, standards, and applications have gained wide acceptance in the Grid computing community. A number of tools and middleware systems have been developed to support application development using Grid Services frameworks. Most of these efforts, however, have focused on the low-level support for management and execution of Grid services, management of Grid-enabled resources, and deployment and execution of applications that make use of Grid services. Simple-to-use service development tools, which will allow a Grid service developer to leverage Grid technologies without needing to know low-level details, are becoming increasingly important. Moreover, support for development of strongly-typed *services, in which data types consumed and produced by a service are well-defined and published in the Grid, is necessary to enable syntactic interoperability so that two Grid endpoints can interact with each other programmatically and correctly. In this paper, we describe an open-source, extensible toolkit, called* Introduce*, to support easy development and deployment of WS/WSRF compliant Grid services. Introduce aims to reduce the service development and deployment effort by hiding low level details of the Globus Toolkit and to enable the implementation of strongly-typed Grid services. We expect that enabling strongly typed grid services while lowering the difficulty of entry to the Grid via toolkits like Introduce will have a major impact to the success of the Grid and its wider adoption as a viable technology of choice, not only in the commercial sector, but also in other areas such as academic, medical, and government research.*

1. INTRODUCTION

Grid computing is quickly becoming the new means for creating distributed infrastructures and virtual organizations for multi-institutional research and enterprise applications. Since its inception, the Grid has evolved from being a platform targeted at large-scale computing applications to a new architecture paradigm for sharing information, data, and software tools as well as computational and storage resources. It has been applied in a wide range of application domains ranging from high energy physics to earth systems sciences to biomedical research.

A vast array of middleware systems and toolkits has been developed to support applications in the Grid [1-28]. These include middleware and tools for application deployment, security (authentication, authorization, and secure communication), remote resource invocation, resource monitoring and scheduling, and workflow management. Earlier tool development efforts, however, focused on providing the core functionality needed by applications to use the Grid environment. As the number of Grid-enabled resources and applications has increased, interoperability has become a major concern, since applications and resources are heterogeneous, oftentimes deployed and managed autonomously, and programmatic access to resources is desired. The need for mechanisms to facilitate interoperability has led to architecture and standardization efforts by several groups, most notably by the Global Grid Forum (http://www.ggf.org), which includes both academic researchers and corporate IT leaders. In the past several years, through a community effort (coordinated via the Global Grid Forum and Oasis), the Open Grid Services Architecture (OGSA) has been developed [29, 30]. OGSA presents a service-oriented view of the Grid and specifies Grid Services standards. These standards adapt and extend Web Services standards[31, 32] for scientific applications. They define mechanisms and guidelines for such additional features as stateful services, service notification, and management of service/resource lifetime. The Web Services Resource Framework (WSRF) standardization effort[33] paved the way for closer interoperability and unification between stateful Grid services and Web services.

With the emergence of Grid Services standards, we are observing the development of new tools and adaptation of exiting tools to leverage the standards in application development and deployment. The most widely used reference implementations of OGSA, as realized by OGSI (Open Grid Services Infrastructure), and WSRF are the Globus Toolkit (GT; http://www.globus.org) version 3.2 (GT 3.2) and version 4.0 (GT 4.0), respectively. The GT enables Grid architects to create Grids using standard building blocks. It also implements support for creation, deployment, and invocation of Grid services, and provides runtime support for common services and tools such as an Index service for service registration and discovery and Globus Security Infrastructure (GSI) for security. These core components enable service providers to stand up Grid services, advertise them, discover them, and secure them. However, developing services using the GT requires a vast knowledge of Grid Services technologies and of the details of the toolkit. We argue that additional tools are needed to enable greater levels of

service-to-service interoperability and to reduce the development time and knowledge required to implement and stand up Grid Services.

In this paper, we present the design and implementation of an open-source toolkit, called *Introduce*, which provides support for development and deployment of *strongly-typed*, secure Grid services. The current implementation of Introduce uses the Globus Toolkit as the underlying core Grid infrastructure and the Mobius framework to support strongly-typed Grid service development. The toolkit aims to reduce the time and knowledge required to stand up Grid services while increasing the interoperability between the services in the grid environment. The salient features of Introduce are as follows:

- It provides high-level functions and graphical user interfaces that encapsulate and hide the common complexities and low level command-line tools of the GT for generating a suitable service layout. These include client and server wrappers to encapsulate the "boxed" grid service calls, functions to create configurable service properties, functions to specify resource properties and register metadata, functions to specify the security configurations of the service operations of the service, and support to deploy a service to commonly used Grid service containers. The service developer can create and modify Grid service interfaces using a graphical user interface (GUI). Developers can also write their own service description documents, which the Introduce toolkit can use to create the service programmatically.
- It enables development of strongly-typed services. A strongly-typed service is one whose methods take as input and return as output data structures that are registered and published in the Grid environment. The use of published types enables a developer to create compatible and interoperable grid services without needing to communicate with other grid service developers. Via the use of plug-ins, Introduce enables discovery of data types from virtually any style of data type providers.
- It is customizable and extensible via the use of *extension plug-ins*. Plug-ins allow for Introduce to be customized and its base functionality to be extended 1) for custom and common service types in an application domain and 2) to employ customized discovery mechanisms for common data types for creating strongly-typed services.
- It allows for implementation of secure services. It leverages all aspects of GSI in order to provide customizable service and method level security configuration and code generation. It

provides support for service developers to optionally turn on authentication and authorization support for individual service methods as well as the entire service itself.

- It manages all the service-specific files and directories required by the GT for correct compilation and deployment. It also generates appropriate, object-oriented client APIs which can be used by client programs to interact with the service.

An earlier version of Introduce is available with the caGrid version 0.5 distribution (https://cabig.nci.nih.gov/workspaces/Architecture/caGrid) [34]; caGrid is the Grid architecture and middleware infrastructure of the NCI-funded cancer Biomedical Informatics Grid (caBIG; https://cabig.nci.nih.gov) program. Originally implemented as a tool to assist analytical service developers, Introduce has evolved into a more generic Grid service development and deployment toolkit. The current version of Introduce described in this paper is in its beta version and has been used as the core caBIG service development toolkit in caGrid version 1.0, which is currently under development. It has been used to build various types of services ranging from data services to core security infrastructure services to biomedical image analysis services. We believe that the availability of tools like Introduce will greatly impact the efficacy of using the Grid in fields where grid style architectures are a great change to the everyday development and management of data and analytical services.

The rest of this paper is organized as follows. Section 2 presents the motivation for the Introduce toolkit. An overview of the related previous work and comparison of Introduce to existing tools is described in Section 3. The Introduce toolkit and its main components, the graphical development environment (GDE) to be used by service developers, the Introduce engine to support the functionality presented through the GDE, and the extension framework that enables customization and extension of the core functionality of Introduce, are presented in Section 4. Finally, we conclude in Section 5.

2. MOTIVATION

The need for the Introduce toolkit is primarily motivated by the cancer Biomedical Informatics Grid program (caBIG, https://cabig.nci.nih.gov), funded by the National Cancer Institute (NCI). The goal of this program is to implement a nationwide cancer research network in order to significantly enhance basic and clinical research on all types of cancer disease. This large scale effort uses the Grid Services architecture as the underlying common architecture.

2.1. EASY DEVELOPMENT AND DEPLOYMENT OF SECURE GRID SERVICES

The principle idea in caBIG is to create software infrastructure, common applications, standards, and common data and analytical resources to leverage combined strengths of individual cancer centers and research labs. In the caBIG effort, a core Grid infrastructure, called *caGrid*, is being developed to support the development and deployment of Grid-enabled applications in order to facilitate sharing of data and analytical resources. The infrastructure is designed as a service-oriented system, building on the WSRF standards[1]. The most widely deployed reference implementation of the WSRF standard is the Globus Toolkit (GT). The GT is hence employed as the Grid middleware backbone and runtime environment in caGrid.

caBIG is envisioned to span several hundreds of institutions[2] and possibly thousands of investigator and research laboratories. These institutions may host information technology infrastructures of different complexity and maturity and consist of users and application developers with varying levels of understanding of distributed systems and the Grid. Thus, one of the goals of caGrid is to provide tools to make it easier for institutions, laboratories, and individual researchers to leverage the Grid without requiring a great deal of knowledge of the Grid technologies. The GT provides support for a suite of core runtime services and tools for developing and deploying Grid services. However, these are low level tools, requiring service developers to understand the details of the GT and how and in what order the tools should be invoked, and to keep track of several files that are required for successful compilation and deployment of services. The Introduce toolkit aims to make it easy for a service developer to execute the various steps of the service development and deployment tools provided by the GT and to manage the necessary directories and files. It abstracts away the details of invoking the various tools so that the developer is freed up to concentrate on the details of implementing his/her domain-specific code.

Security is a required component in caBIG both for protecting intellectual property and to ensure protection and privacy of patient related information. In caBIG, a service provider should be able to enforce secure and controlled access to resources based on policies set by the owners of the resources. caGrid provides higher level services such as Dorian [21] and the Grid Trust Service (GTS) for managing and enforcing security. Introduce allows a service developer to

[1] The initial release of the caBIG architecture, called caGrid version 0.5, is built using the OGSA standards and Globus Toolkit 3.2. The new release, caGrid version 1.0, is being developed using the WSRF standards and Globus Toolkit 4.

[2] There are currently more than 50 cancer centers participating in the caBIG effort. (https://cabig.nci.nih.gov/participants)

configure the service as a *secure* service. This enables the service to interact with Grid-wide and local authentication and authorization services and systems.

2.2. STRONGLY-TYPED SERVICE DEVELOPMENT

The caBIG effort also aims to enable syntactic and semantic interoperability across data and analytical resources. The OGSA and WSRF frameworks enable interoperability by defining standards for data and information exchange protocols, service creation and management, and service invocation. In the Grid environment, syntactic and semantic interoperability is important, when data is exchanged between disparate groups of data providers and consumers. It is critical to ensure that resources can be accessed programmatically in a unified way and the information served by a resource can be interpreted correctly. In the Mobius project [20, 35], we introduced the notion of *strongly-typed services* in the context of virtualization of data sources to enable syntactic interoperability. The Mobius framework provides tools and services for coordinated management of XML schemas; a schema defines the data types and structure of a data object. We refer to a service as strongly-typed service, if the input and output parameters of the service methods conform to schemas[3], which are well-defined and published in the Grid environment. Similarly, in caBIG, syntactic and semantic interoperability is accomplished through coordinated management of metadata and well-defined objects, bound to underlying semantic concepts, which are exchanged between entities (i.e., between services, or between clients and services) in the environment. The structure of a registered object is represented by a registered and published XML schema. The objects are exchanged between two end points in the Grid as XML documents conforming to the corresponding schema.

Introduce enables the developer to implement *strongly-typed* services, focusing on syntactic interoperability. It provides the required support using the Mobius Global Model Exchange service. In domains with publishes semantic ontologies, such as caBIG, semantic interoperability can also be facilitated through the use of ontology aware data type discovery extensions.

3. RELATED WORK

Most of the previous work on tooling for Grid computing targeted the development of basic, low level functions required to stand up a Grid service, manage resources, and execute applications,

which interact with Grid services[1-28, 36-38]. Our work differs from previous low level middleware development efforts in that it aims to provide a high-level toolkit for development and deployment of strongly-typed, secure Grid services.

The Globus Toolkit (GT; http://www.globus.org) and related technologies such as Axis (http://ws.apache.org/axis/) and Java Naming and Directory Interface (JNDI; http://java.sun.com/products/jndi/) are the cornerstone pieces leveraged by the Introduce toolkit. The GT is a software package which, in addition to aiding in the development of WSRF compliant Grid services, provides a suite of common services (e.g., the Index Service and MDS (Monitoring and Discovery System) components for service advertisement and discovery) to create a Grid environment. Apache Axis is an open source middleware providing the actual Web Services layer, i.e., SOAP based Web Service protocol[32] and support for Java bean generation, resource support via JNDI, and other Web Service middleware components. Introduce uses the GT, Axis, and JNDI as the underlying middleware systems to support development of stateful services, advertisement of metadata in the form of resource properties, and deployment of services for execution.

The Globus Service Build Tools Project (http://gsbt.sourceforge.net/) consists of GT4IDE and globus-build-services components. The GT4IDE component is an Eclipse (www.eclipse.org) plug-in designed to aid in the development of Globus Toolkit 4 (GT4) compatible services -- the predecessor of GT4IDE, the GT3IDE component, supports services compliant with the Globus Toolkit version 3 implementation. The plug-in enables a service developer to create a service and add methods to the service. It then generates the stub service to be implemented by the service developer. Unlike the Introduce toolkit, which is a stand-alone application, the GT4IDE plug-in can only be used in the Eclipse environment. While the plug-in offers a great tool for developers, who are experienced in Grid technologies and the GT, it lacks several features found in Introduce. These features include support for strongly-typed services, removal of document literal bindings to the user defined service interface, extensibility, auto-code generation for common interfaces, and ability to leverage advanced features of GT4 such as resource properties framework, compositional inheritance, and service and method level security configuration. Moreover, GT4IDE mainly facilitates development of server side capabilities, whereas Introduce can be used to generate both server side and client side methods and APIs. The globus-build-

[3] A schema may represent a simple data element such as integer or a more complex object.

services component is an Ant build file and script to generically compile the grid service and generate an appropriate GAR file. Introduce leveraged a modified a version of the globus-build-services for use in the services that it generates.

Morohoshi and Huang[39] propose a toolkit designed to support service development on top of the GT3 platform, which is based on the OGSA standards. An application developer can use the graphical user interface of the toolkit to create a basic service by inputting a set of parameters or a service from a template, generate service stub from GWSDL (Grid Web Service Description Language) specification, generate GWSDL from service method interfaces (implemented in Java), set service metadata, build necessary files, and deploy or undeploy the service. The toolkit can generate client packages for a service as well. Mizuta and Huang[40] develop a prototype cartridge for AndroMDA (http://www.andromda.org), an open-source model driven architecture (MDA) based code generation framework, to facilitate GT3-compliant service development using UML. In their framework, Grid services are expressed as class diagrams and models in UML. The XMI file generated from the UML diagrams via a tool like Enterprise Architect or Poseidon is processed by the AndroMDA cartridge to generate the source code files and associated settings. Their approach allows a service developer to use existing UML-based modeling systems to design Grid services. It also makes reuse and maintenance of designed services easier, since models based on MDA are platform and language independent. The cartridge implemented for the AndroMDA implements the necessary functions and extensions so that all the files, directories, source codes, and service data elements are generated for a service being developed. One limitation of their work is their tool can be used only in the AndroMDA framework.

These two works and Introduce share common goals; they aim to make it easier to develop Grid services by providing higher-level functions and easy-to-use user interfaces. However, Introduce has several unique features not available in the previous toolkits. These features include 1) a more comprehensive extensibility framework, which goes beyond just providing service templates and allows complex plug-ins (providing both logic and custom graphical components) to be integrated to the toolkit, 2) support for strongly-typed services, and 3) ability to leverage resource framework of GT4 and compositional inheritance. On the other hand, Introduce could use a component like AndroMDA and the cartridge as an extension plug-in in

the Introduce extension framework (see Section 4.6 for more information on the extension framework) so that a service can be designed using UML models.

The IBM Grid Toolbox (http://www-1.ibm.com/grid/solutions/grid_toolbox.shtml) is a suite of integrated tools to aid with creation and hosting of Grid services using the GT3 platform. It includes tools developed by the Globus project and a set of additional APIs and tools for service creation and deployment. It also provides a suite of common services, both from the open-source Globus Toolkit as well as services implemented by IBM to support common operations in a Grid environment. Unlike Introduce, which provides access to service development and deployment functions from a graphical user interface, the IBM Grid Toolbox Software Development Kit (SDK) consists of command-line tools (e.g., Ant task and XML batch file based tools) and APIs that have to be invoked individually by the service developer. In addition, the toolbox does not provide support for strongly-typed service development.

The Java CoG kits[41, 42] offer a rich set of client side capabilities to develop clients and access Grid functions and services. Only limited functionality is provided for server side capabilities and service development. Introduce implements support for development of both clients and services. The Gridbus project[43](http://www.gridbus.org/) provides a set of tools for Grid-enabled application development, in addition to components for economy driven scheduling of resources and applications, simulation of Grid environments, workflow composition and execution. The WSRF .NET[44] is a project to develop a reference implementation of the WSRF standards on the .NET platform. It supports creation of services using the ASP.NET and Visual Studio .NET environments and a set of WSRF .NET tools (e.g., the PortTypeAggregator tool for service deployment). The tools are mainly targeted at service side development and provide limited functionality for client side implementation. The Introduce toolkit, on the other hand, is a stand-alone program and consists of self-contained suite of components targeted at service development on the GT4 platform.

The work presented by Smith et. al. [45] lays out the importance of a model driven architecture (MDA) approach to generating Grid service oriented architectures. Their work primarily deals with creating a service generation system for GT4 which uses an MDA along with a code annotation approach. They propose an architecture for generating Grid services using Java annotations. The user can create the Java service implementation and simply annotate the methods and resources and their components with use Java Annotations to process the

service implementation and create the Grid service. This work is a nice concept for easily separating the business logic of a service from the Grid service implementation; however, it does not enable the user to clearly stay away from the complexities of the underlying toolkits for such things as security, registration, invocation, or composition. The MDA concepts, however, do hold true in the Introduce toolkit. The separation in layers between the business logic (Code Layer), platform binding (PSM), and overall conceptual design (PIM) are very relevant to the design philosophies exposed by the Introduce Toolkit.

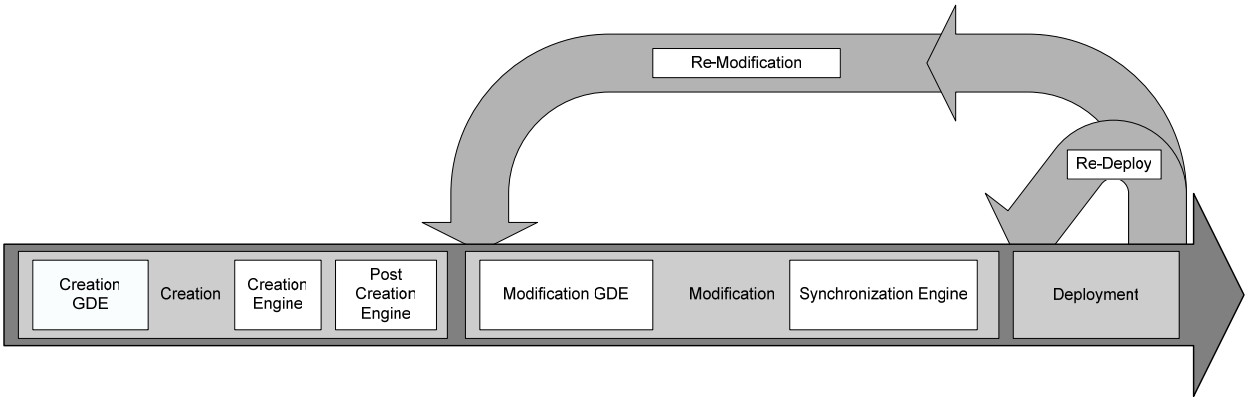


Figure 1. Overall service development process.

4. INTRODUCE TOOLKIT

The Introduce toolkit is designed to support the three main steps of service development (Figure 1): 1) *Creation of Basic Service Structure.* The service developer describes at the highest level some basic attributes about the service such as service name and service namespace. Once the user has set these basic service configuration properties, Introduce will create the basic service implementation, to which the developer can then add application-specific methods and security options through the service modification steps. 2) *Service Modification.* The modification step allows the developer to add, remove, and modify service methods, properties, resources, service contexts, and service/method level security. In this step, the developer creates strongly-typed service interfaces using well-defined, published schemas, which are registered in a system like the Mobius GME, as the type definitions of the input and output parameters of the service methods. 3) *Deployment.* The developer can deploy the service which has been created with Introduce to a Grid service container (e.g., a Globus or Tomcat service container).

A service developer can access the functions required to execute these three steps through the Graphical Development Environment (GDE) of Introduce. The runtime support behind the GDE functionality is provided by the Introduce engine, which consists of the Service Creator, Service Synchronizer, and Service Deployer components. The toolkit provides an extension framework that allows Introduce to be customized and extended for custom service types and discovery of custom data types. In the following sections, we describe the GDE, the Introduce engine, and the extension framework in greater detail.

4.1. EXTERNAL MIDDLEWARE SYSTEMS AND TOOLS

Introduce assumes the availability of two external components to implement its functions. 1) A Grid runtime environment that provides the support for compiling, advertising, and deploying Grid services; 2) A repository of data types that is accessible locally or remotely so that the toolkit can pull the common data types for strongly-typed services. The current implementation of Introduce leverages several open source middleware systems such as the Globus Toolkit (GT), Apache Axis, and the Mobius Global Model Exchange (GME) service. However, it is designed as a modular system, in which the specific implementations of external components can be replaced.

The GT and Apache Axis implement the core Grid/Web Service support. The Introduce toolkit uses the GT as the underlying Grid runtime environment. It generates the service layout, the service description (WSDL), and the service files such that they are compliant with the GT and Axis and can be readily deployed.

Introduce uses the Mobius GME service[20] as a repository of XML schemas to support the development of strongly-typed services; it also has the ability for custom data type discovery plug-ins, detailed later. The GME is one of the core services provided in the Mobius framework. It implements support for coordinated management of XML schemas in a distributed environment. In GME, all schemas are registered under namespaces and can be versioned. A schema can be a new schema, or can contain pointers to other registered schemas and the attributes of schemas, or can be generated by versioning an existing schema (e.g., by adding or deleting attributes). The concept of versioning schemas is formalized in GME; any change to a schema is reflected as either a new version of the schema or a totally new schema under a different name or namespace. In this way, data types can evolve while allowing clients and

services to make use of old versions of the data type, if necessary. Namespaces enable distributed and hierarchical management of schemas; two groups can create and manage schemas under different namespaces without worrying about affecting each other's services, clients, and programs. In the Grid environment, data elements and objects are exchanged as XML documents conforming to a schema. In such a setting, the schema describes the structure of a simple or complex data element (data object) that is consumed or produced by a service and is exchanged between two endpoints in the environment.

4.2. INTRODUCE GRAPHICAL DEVELOPMENT ENVIRONMENT

The Introduce Graphical Development Environment (GDE) is the graphical user interface that can be used to create, modify, and deploy a service (see Figure 2). It is designed to be very simple to use, enable using community excepted data types, and provide easy configuration of service metadata, operations, and security. It also allows customized plug-ins to be added for such things as repositories of data types and for creating custom or common service types.

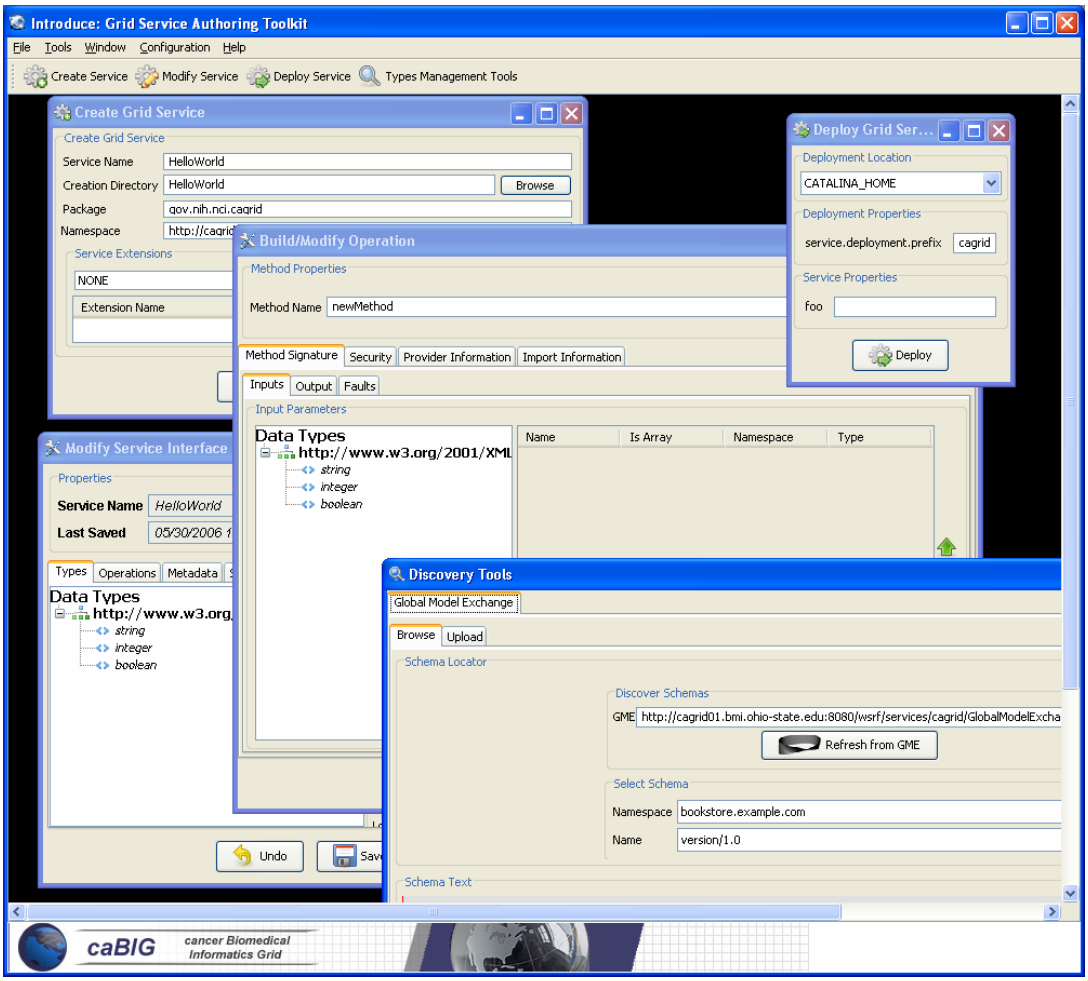


Figure 2. The Introduce Graphical Development Environment (GDE).

The interface contains several screens and options for the service developer to 1) create a new service, 2) modify an existing service, 3) discover and use published data types in order to create strongly-typed service methods, and 4) configure service metadata and deploy the service.

4.2.1. SERVICE CREATION

The service creation component, shown in Figure 3, enables the developer to create a new service. Using the interface, the service developer can provide some basic information such as the name of the service, package name to use for creating the source code, and domain name to use in service description (specified in WSDL), when defining the methods of the service.

The developer also has the ability to add service extensions. A service extension is an Introduce plug-in (see Section 4.6 for more details), which is designed to add custom settings on the service. For example, service extensions might add pre-defined operations,

resources/resource properties, or security settings. They enable the development of custom service types with predefined methods, which must be implemented. They also enable Introduce to run the custom code implemented in the plug-in, which makes modifications to the underlying service being created. This capability allows the specialization of Introduce to support domain specific common scenarios, further abstracting the individual service developer from responsibilities related to the deployment of grid technologies in a production environment.

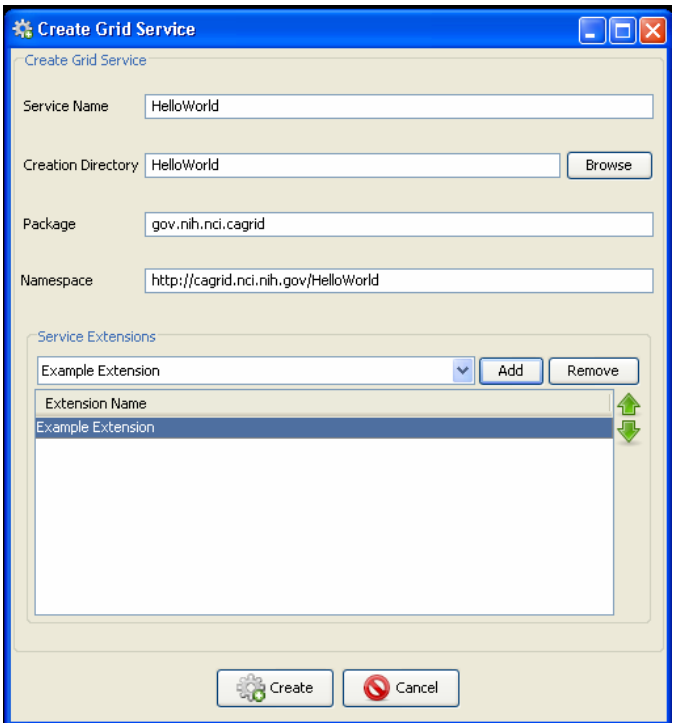


Figure 3. Introduce GDE Service Creation Component.

4.2.2. SERVICE MODIFICATION

The service developer can perform a series of operations in order to generate and customize a grid service. First, the developer needs to discover the data types that are desired to be used as the input and output types of methods of the service and the data types for describing the resource properties of the service. This is done via the "Types" tab shown in Figure 4 (a) of the GDE Service Modification interface. This tab shows the current types the service is using, and provides access to the data type discovery components (such as the Mobius GME), for selecting and configuring additional types. Second, the developer can add, remove, or modify operations on the service, using the "Operations" tab shown in Figure 4 (a). For each operation, the developer needs to set the input parameters, return type, any fault types that can be thrown from

each service method. The security configuration of the service method should also be set. The respective GUI components for these options can be seen in Figure 4 (b).

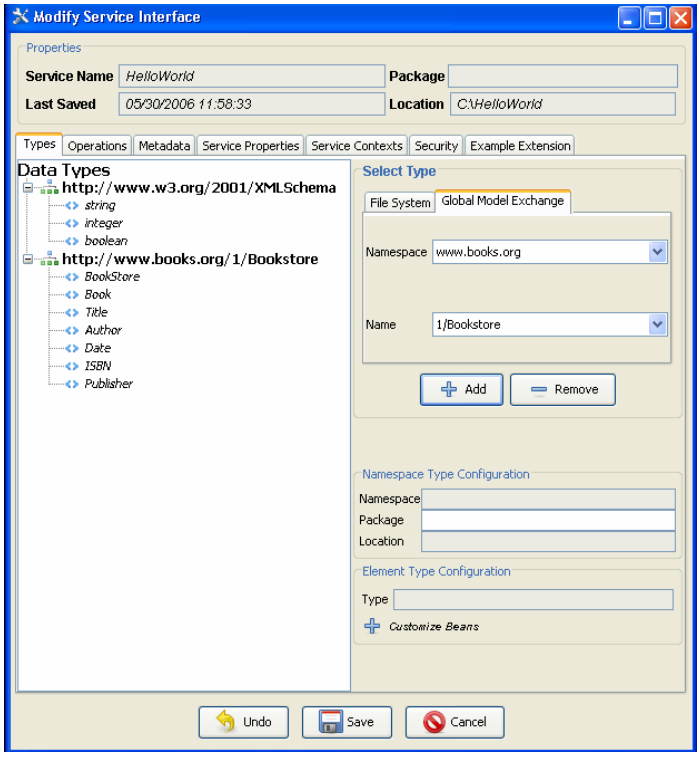
Service methods can also be *imported* from other services. For example, if a developer would like all services or a certain type of service to implement a particular method, the corresponding method can be imported from another service. The importing of a method across services will assure not only that each service has completely protocol compatible methods but also that each service's method can be invoked by the same base client. This enables the notion of basic inheritance in grid services and will be discussed further in Section 4.3.

Service state information and metadata in the form of resource properties can be added, removed and configured via the "Metadata" tab shown in Figure 4 (a). This facilitates such operations as registration with an Index Service, and configuring if the values should be loaded from files or populated by the service implementation at runtime. Service properties, which are set at deployment time and passed into the service, can also be added and their default values can be set via the "Service Properties" tab shown in Figure 4 (a).

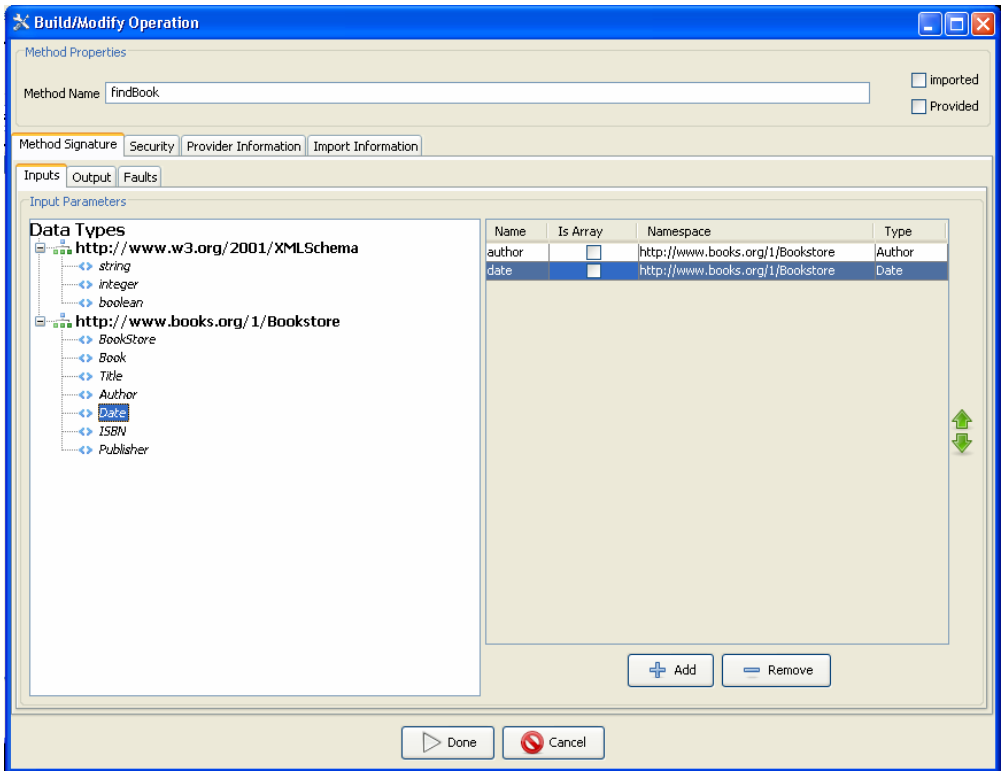
Security on service can be configured via the "Security" tab show in Figure 4 (c). Security can be configured both at the service level and at the method level; method level security takes precedence over service level security. Methods that don't specify security will inherit the service level security configuration. The "Security" tab contains several sub tabs, "Secure Communication", "Authorization", "Client", and "Service Credentials". The sub tabs allow configuration of various security properties enabling the developer to easily configure custom security models. The "Secure Communication" tab allows the configuration of the acceptable secure communication mechanisms supported by the service. Introduce allows the configuration of all three secure communication mechanisms supported by the grid, Transport Level Security (TLS), Secure Conversation, and Secure Message. The "Authorization" tab enables the configuration of authorization mechanism(s) used in restricting access on the grid service. The "Client" tab enables the developer to configure security on the Introduce generated Clients. This includes specifying client authorization, the use of anonymous access, delegation mode, and the communication mechanism to use (if the service supports multiple mechanisms). The "Service Credentials" tab enables the developer to specify a set of credentials for the service to use for authenticating with clients and other services. Credentials may be specified as a certificate and

private key or a grid proxy. If no credentials are specified the service will inherit the container credentials.

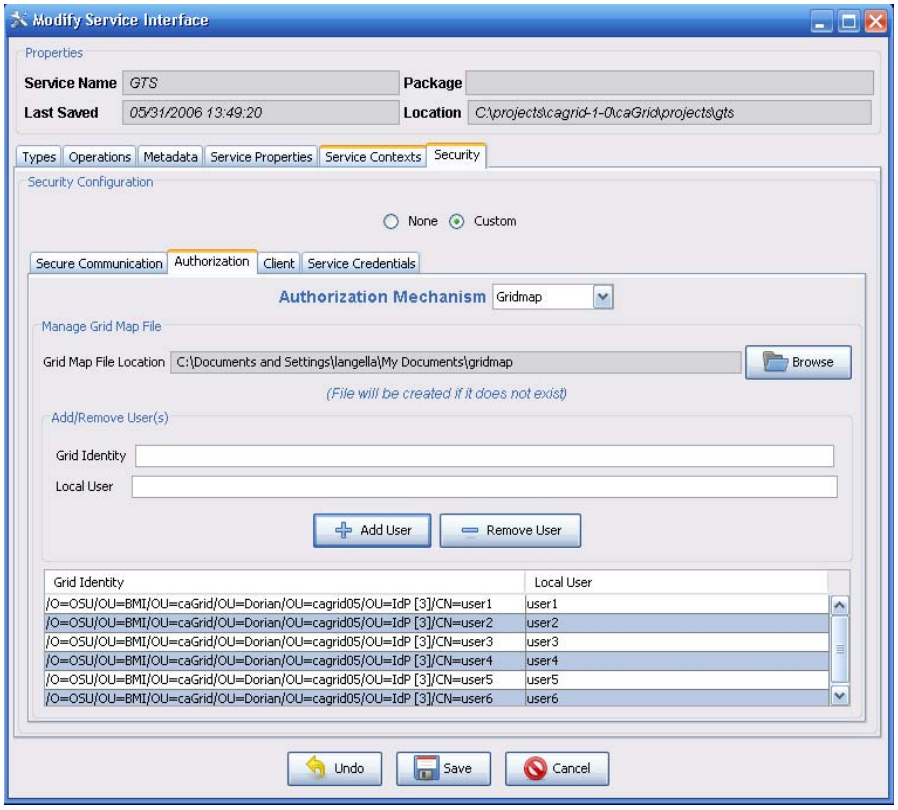
The last feature which is enabled at modification time is the addition or removal of service contexts, via the “Service Contexts” tab shown in Figure 4 (a). Service contexts define additional conceptual contexts of operation needed to support the desired service functionality. As an example, if an operation on the main service enables the user to query a database, that operation might create a resource in another context and return the handle to that context to the user as opposed to the full query result set. This secondary context can then enable the user to iterate through the query results. This is accomplished by operations or resource properties to this secondary service context which will be responsible for iteratively giving results to the user. It should be noted that multiple instances of these contexts can be created and executed concurrently; one for each query that comes in, for example. This style of grid service is supported by the WSRF specifications. Though the details of the WSRF-implementation of these concepts are abstracted away from developers its worth noting how they are realized, and this is described in Section 4.4.4. Introduce makes it easier for service developers to create such complex services, via the GDE, without having to fully understand the underlying service implementations.



(a) Modifying overall service design.



(b) Modifying service operations.



(c) Configuring Security.

Figure 4. Introduce GDE component for creation and modification of a service operation.

4.2.3. DISCOVERY TOOLS: SUPPORT FOR STRONGLY-TYPED SERVICE METHODS

Using the GDE, developers can obtain the types that they want to use for the service parameters and return types from any data type discovery plug-in. Utilizing common and standard data types, which are defined outside of any application-specific service, enables the creation of strongly typed grid service interfaces. This increases service-to-service interoperability. Once a data type is selected through the GDE, the data type is retrieved, written into the schema area of the service, and imported for use in the service WSDL description so that Java beans can be generated and the data types can be programmatically used.

The Introduce toolkit comes with a set of pre-installed discovery plug-ins, such as the Mobius GME and a basic file system browser which can be used to locate local schemas. The GME plug-in enables developers to browse, upload, and download schemas published in a GME. These schemas represent the valid data types which can be used during service creation. Using the GME plug-in, a developer can take a schema, create an editable view of the schema, and then

submit the schema to the GME. If the namespace of the schema is not managed by the GME, to which the schema is submitted, the plug-in will attempt to add the namespace to the GME before submitting the schema. One or more GME services can be available and accessible in a Grid environment[4]. Once the schema has been uploaded to a GME in a given Grid environment, it can be discovered and used by users in the same Grid environment through the Introduce toolkit. The GME plug-in browser window enables browsing through all the GME published types by namespace and schema name. It gives the user a quick view of the schema and the option to download the schema bundle. The schema bundle contains the schema and all other schemas which are referenced by that schema.

4.2.4. DEPLOYMENT

The deployment option of the GDE (Figure 5) allows the service developer to deploy the implemented grid service, which has been created with Introduce, to a Grid service container. The toolkit currently supports deploying a service to either a Globus or Tomcat Grid service container; however, support for other deployment options can easily be added to the GDE. The deployment window allows the service deployer to populate service configuration properties, which the service will have access to at runtime. Then the service is deployed to the selected container.

[4] The GME infrastructure is similar to DNS (Domain Name Servers) in that multiple GMEs can be organized into hierarchies or peers.

Figure 5. Introduce GDE service deployment component.

4.3. INTRODUCE ENGINE

The runtime support to enable service creation, modification, and deployment is provided by the Introduce engine. In this section, we describe the main components of this engine.

4.3.1. SERVICE CREATOR

The service creator is composed by a series of templates using the Java Emitter Templates (JET) component, which is part of the Eclipse Modeling Framework (http://www.eclipse.org/emf/), for generating source code and configuration files, and a skeleton set of directories which is used to generate a Grid service that can be built, registered, and deployed in the Grid environment.

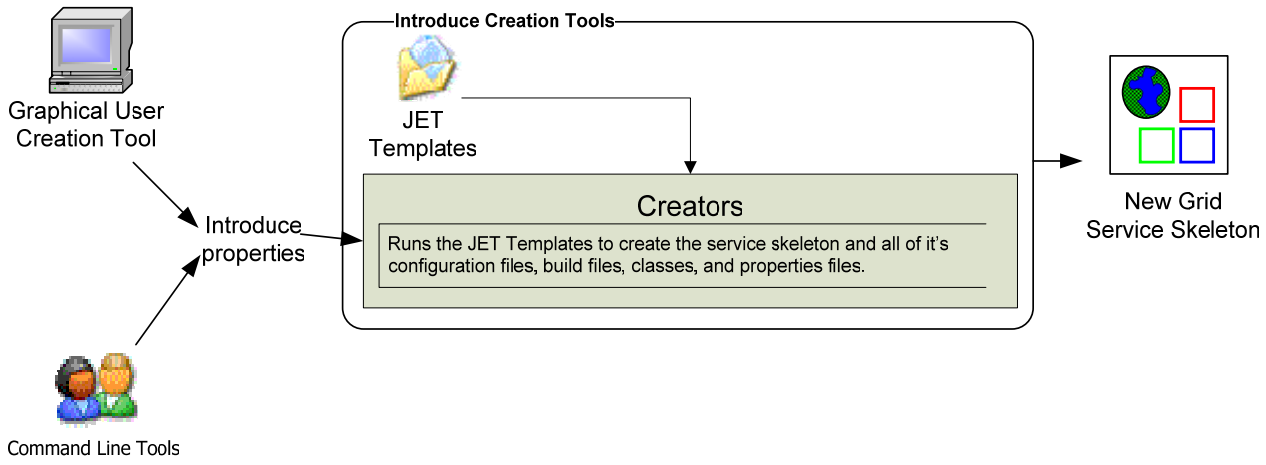


Figure 6. Service creation tools and use of JET templates for service creation.

Templates for source code and configuration files are used to create all the custom Java source code for the service and the client APIs, and to generate the files required by the GT in order to build and deploy a Grid service (see Figure 6). Deployment configuration files are used for resource and resource property configuration in the form of Java Naming and Directory Interface (JNDI), resource property registration configuration, service deployment descriptor in the form of Web Service Deployment Descriptor (WSDD), and security configuration. The basic service created by Introduce has the following components:

- Ant processes for build, deploy, and test operations,
- custom configuration files for IDE integration, e.g., Eclipse project files for editing of the service using the Eclipse platform (www.eclipse.org),
- standard interface for both client and service to implement,
- fully implemented client APIs,
- stub implemented service,
- configuration to support service metadata and resource properties and the registration of metadata and properties,
- configuration for secure service deployment and authorization.

Examples of the various files and directories managed by Introduce for a service are illustrated in the Appendix.

4.3.2. SERVICE SYNCHRONIZER

Service resynchronization is the process by which the source code and configuration generation tools of the Introduce toolkit will analyze the service's current implementation with that of the desired service description. This process will add, remove, and modify any service methods, resource properties, and service settings which have been added, removed, or modified from the service description. The descriptions and configurations for methods, metadata, and security are those that are generated from the GDE, programmatically or by hand, and that can be validated by the Introduce service schema (Figure 8 and Figure 9). The service description is the basis by which the code generation tools add, remove and modify operations, metadata, and change the security configuration of the service by editing the source code, configuration files, and metadata files. This is similar to the way that Axis will use the WSDL to generate the grid stubs of the service. The overall high level process of service resynchronization is illustrated in Figure 7.

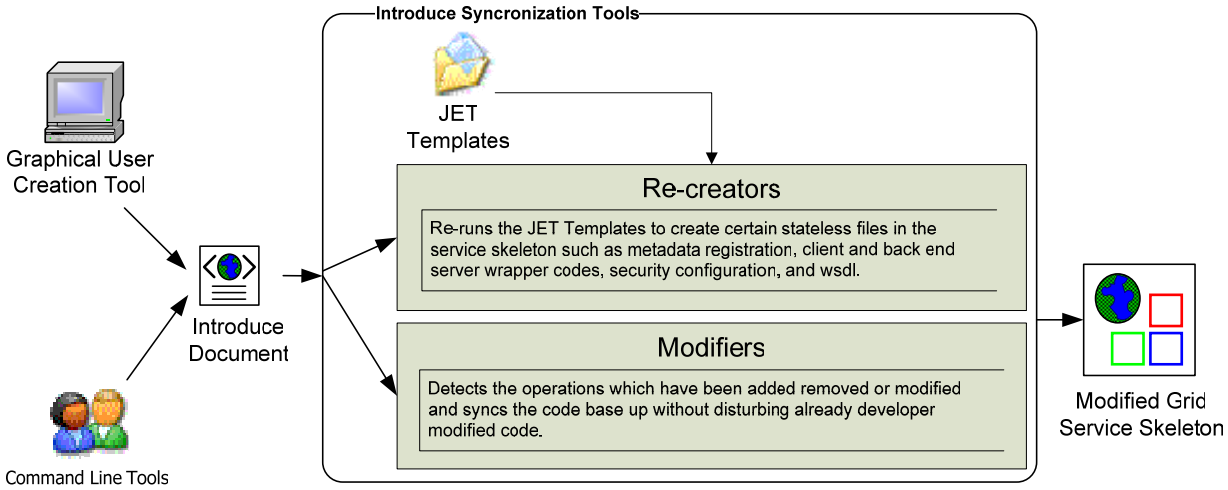


Figure 7. Service resynchronization process.

In this process, modified documents created by the GDE are processed using JET templates to create the modified service skeleton. The service synchronizer component manages the service WSDL description. If a service or method has been added or removed, the respective WSDL files must be updated to reflect the changes. Updating the files requires many auto generated code segments. External XML schemas, which describe published data types, must be imported into the respective WSDL files so that the data types can be located and used to generate the required Java beans and SOAP bindings. The WSDL description is basically the Grid layer

representation of the Java based service interface. Both the *message* types, which represent the data types of the input parameters and the return type of a service method, and the complete *operation*, which reflects the Java based signature of the method being exposed, have to be described in the WSDL file. The synchronization operation automatically keeps this file in sync during the development of services so that the service implementation and the Grid service description represented by the WSDL match up. This ensures that the methods implemented in the Grid service can actually be invoked.

The service can have an inheritance model by adding methods from another service, possibly along with the implementations of those methods (see Section 4.4.3). If a method were imported from another Grid service, the service synchronizer component would also pull in the WSDL description of the method and copy it into the portType of the new service. This enables the service to have a completely protocol compatible implementation of the method.

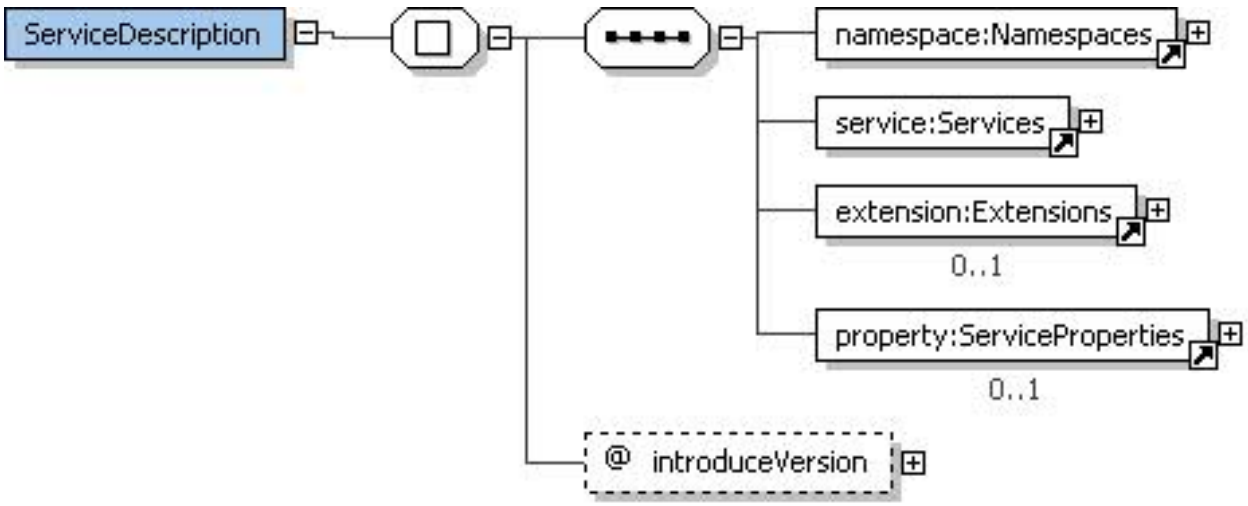


Figure 8. Base Introduce Service Description Schema

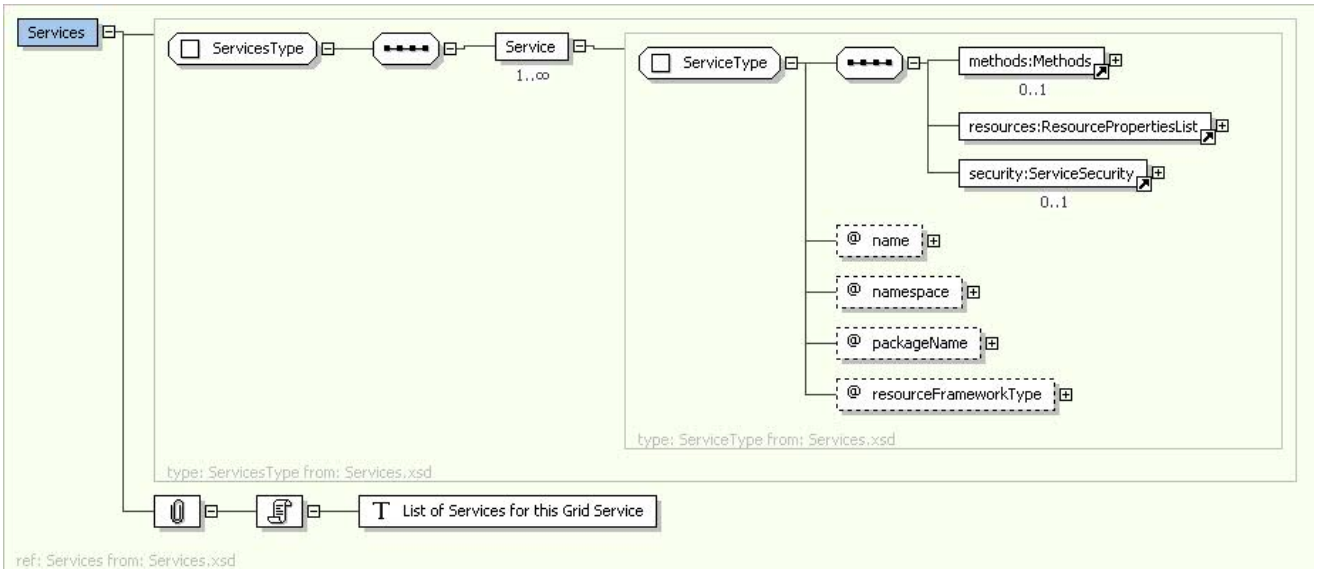


Figure 9. Schema for Introduce Services.

4.3.3. SERVICE DEPLOYER

The service deployment features of Introduce currently support deploying Grid services to a Globus or Tomcat container. The deployment framework can easily be extended to provide deployment capability to other Web Service containers. It utilizes the layout of the Grid Archive (GAR) structure to organize and package a deployable service. Once the GAR for a service is generated by the deployment framework, the GAR can be utilized to deploy the service to the appropriate container. The deployment framework also allows for deployment time service properties to be acquired from the application developer or the user deploying the service. These custom service properties allow a service creator to define configurable options that make sense for a particular deployment of the service to have control over. When the service is deployed, the deployer prompts for the desired values, presenting service developer provided defaults. The supplied values are then made available to the service at runtime. For example, if a service implementation accesses a local database, the username and password of the database can be specified as service properties. This allows a very convenient way for those deploying the service to configure it appropriately for their environment, without requiring knowledge of how the grid service implements these configuration points.

The deployment process first populates files required in deployment (e.g., the WSDD and JNDI of the service) to specify such values as service deployment path or security configuration file locations. It then gathers the library files required for the service as well as those library files

which contain the actual runtime code of the service. All configuration files and service resources are also collected. Finally, a GAR file is generated for this service. Once the GAR file has been generated, it can be handed off to the particular deployment handler for the required container to be used for this deployment.

4.4. OTHER FEATURES OF INTRODUCE ENGINE

4.4.1. AUTO-BOXING/UNBOXING OF SERVICE OPERATIONS

Grid service code generation and synchronization is one of the most complicated code generation operations in the toolkit. This component has to re-write the basic source code of the service so that the client and the server both agree on the new modified interface as designed by the service creator. It also has to deal with associating the server side implementation of the service designer's interface to the actual port type generated by Globus via Axis. The overall process is complicated due to fact that GT and Axis use document literal bindings to create the services portType and bindings to SOAP. For example, if a user describes a new method as shown below:

```
int foo(int bar1, int bar2);
```

The port type method that will be created for the corresponding Java interface call will look like the one below:

```
FooReturn foo(FooParameters params);
```

This style is known as document literal binding. This boxing or wrapping of the parameters and the return type of the service method can be confusing to the service user and service developer to deal with, especially since this document literal style is exposed directly through the client API or the service implementation API. Every client using the service will have to box up the parameters to call the operations of the service and un-box the results. Not only will this task be cumbersome for service users, but the document literal interface is not the interface that the service designer intended to be provided to its users. The Introduce toolkit will hide the boxing and un-boxing of methods by providing an interface to the service, which looks exactly as described by the service developer and not as interpreted by Globus via Axis. In order to do this, the toolkit creates a wrapping layer in the client and service which both implement the clean interface (non document literal). These wrapper layers auto-box and un-box and map the calls

from the clean client to the document literal port type client generated by Globus/Axis and visa versa for the service. It is worth noting that the move to document literal binding was made in the GT for interoperability reasons, as the more developer-friendly APIs provided by Introduce, and previously provided by GT, are not standardized and that Introduce created services can still be accessed via the standard document literal interfaces.

4.4.2. RESOURCE FRAMEWORK

Introduce provides support for exposing service state and metadata by abstracting away the details of common patterns of use of the WSRF. The WSRF specifications and the GT implementation of these specifications allow the creation of stateful Grid services, whilst remaining compatible with existing Web Service standards. The state (and metadata) of a Grid service is maintained in *Resources* in the WSRF specification, and is exposed to clients by way of *Resource Properties*. Introduce allows its users to expose state or metadata of their services by managing the definition, creation, and population of the Resources and Resource Properties of a service.

Upon creating a service, a service developer is able to simply select from a list of common resource usage patterns, and Introduce manages the complete generation of the necessary backend code and configuration to implement that pattern. Once the resource pattern is defined, a service developer can then use the existing data type discovery tools (as used to specify operation inputs and outputs) in order to expose state or metadata of its resources by way of Resource Properties. Developers need only select a data type they wish to expose, and Introduce makes it available as WSRF standard Resource Property. Developers are able to either programmatically control the values of the property at runtime (as is common for exposed resource state), or supply the value from a file at service startup (as is common for exposed metadata). Another powerful feature of Resource Properties is that they can be advertised to, and aggregated at, a remote indexing service – an indexing service can be used by clients and other services to discover services in the Grid environment. GT provides such a service in the form of the Index Service. It acts as the white and yellow pages of the Grid, providing resource and service discovery.

Introduce allows service developers to maintain soft-state registration of resource properties simply by selecting a check box on each property they wish to register. For each such property,

it creates all of the necessary source code and configuration necessary to periodically register the property with an Index Service, and ensure the most recent value of the property is made available in the Index Service and associated with the service. The combination of these two simple-to-use, yet powerful features, allows a service provider to, for instance, automatically provide and register a description of its service to a central repository when the service is running.

4.4.3. COMPOSITIONAL INHERITANCE

Introduce enables the service developer to import methods from other services (i.e., from other portTypes). This notion is called *compositional inheritance*. Services under the new WSDL 2.0 specification cannot extend portType definitions. In order to simulate portType extension Introduce implements the ability to copy *operation* descriptions from other portTypes and put them in the service's own description. When using this feature, there are various configuration option the engine must know such as the namespaces of the operations, and whether or not an implementation of the operations are already provided or should be generated stubbed just as any other Introduce defined operation.

In order to be sure that the operation can be correctly imported and copied into the new portType, the WSDL of this operation and any referenced schemas must be brought into this new service and imported. If the operation implementation is not being provided, the synchronization engine must add this operation to the services base interface, and the un-boxing Globus wrapper for this operation must be generated. If the implementation of this operation is being provided, the extra code in the interface, its implementation, and the wrapper do not need to be added. However, the implementation code, in the form of a JAR file, must be brought into the new service, and the operation class must be added to the WSDD of this service.

4.4.4. MULTI SERVICE / MULTI RESOURCE

In grid services architecture it is sometimes required that a service not only maintain some extra information about the service, but also maintain information (state) which is of particular interest to one particular user. These styles of Grid service use cases have driven the requirements for specifying a mechanism for stateful Grid services. Each WSRF service manages its state by creating and manipulating Resources. A Resource can essentially be thought of as an arbitrary

state representation, as defined by the service developer. The main restriction is that a given service can only manage a single given resource type.

As previously stated in section 4.2.2, Introduce supports this mechanism through the *Service Context* concept. For example, a Data Service may want to define two contexts: the main "Query" context, which provides the ability to query into a backend database, and the secondary "Results Delivery" context, which provides the ability to iteratively access query results (similar to a remote cursor). Introduce implements these Service Contexts by creating a WSRF service for each context (i.e., a Query Service for the Query context and a Results Delivery Service for the Results Delivery context), and a corresponding Resource type. In this example, the Query Service would have a Resource type that represents the backend database(s), and the Results Delivery Service would have a Resource type that represents query results. When a client submits a query by invoking the *query* operation on the Query Service, the service can query the backend database and then create an instance of the Results Delivery Service's Resource. The *query* operation then returns a pointer, or *EndpointReference* (EPR), to this Resource. The client is then able to interact with the results of the query via the Results Delivery Service's client API, passing in the returned EPR. Through its support for multiple Service Contexts, Introduce enables this and other such Resource patterns for stateful Grid services.

This support is managed by the synchronization component of the Introduce engine. The service description model, created via the GDE (or programmatically/hand generated), enables the description of multiple Service Contexts in an Introduce service. Each Introduce service has at least one Service Context (the main service), and can create an arbitrary number of additional Service Contexts to support more complex resource usage patterns, as described above. Each created context has a corresponding source directory containing its own server, client, common, resource, etc. This enables resource properties, operations, and security configurations to be added, removed, and modified for each additional context. Each additional context's corresponding service and resource are modified, compiled, and deployed with the main service.

4.5. SECURITY

Introduce facilitates the creation and configuration of secure Grid services using the Grid Security Infrastructure (GSI) and allows security to be configured at both the service level and the service method level. Moreover, Security can be enforced on both the client and service

sides. Introduce allows service developers to specify the secure communication mechanism(s), in which clients are allowed to communicate with the service. An Introduce generated client can automatically be configured to communicate over the secure communication mechanism specified by the service. In the case where multiple secure communication mechanisms are supported by the service, Introduce will allow the service developer to choose which mechanism the client will use.

Grid Services often need credentials such that they may authenticate with one another or so they may authenticate with clients. Depending on the communication mechanisms supported and the deployment scenario, a service may inherit its credentials from the container hosting the service, or it may be configured to have its own credentials. The GDE facilitates the configuration of service credentials. Service credentials can be configured in the form of certificate/private key or in the form of a Grid proxy. Introduce also facilitates of additional client security aspects, these include anonymous secure communication and delegation.

The GDE allows for the configuration of both client side and service side authorization. Clients can be configured to perform Self Authorization, Host Authorization, or Identity Authorization, on a Grid service before allowing the Grid service to be invoked. Client side authorization is done based on the service credentials presented to the client by the service. On the service side, Introduce allows the configuration of the Globus Authorization Framework, which enforces authorization policy configured on services.

At synchronization time, the Introduce engine interprets the security configuration and uses it to configure the service accordingly. Configuration of service security requires the engine to make modifications to the client source code, service source code, security descriptor, and server deployment WSDD. The client source code is modified to configure the generated stubs to use the appropriate secure communication mechanism, enforce the specified client authorization mechanism, use the delegation mode specified, and whether or not to communicate anonymously. The service source code is modified to support the delegation code specified. A security descriptor is created to specify the secure communication mechanism supported by the service, configure the authorization framework, and to configure the service credentials. Finally the server deployment WSDD is edited to add any configuration parameter that may be needed.

4.6. INTRODUCE EXTENSION FRAMEWORK

The Introduce Extension framework currently consists of two styles of extensions; service and data type discovery. These extensions are implemented by service or data type discovery components which add custom functionality to the Introduce framework. Extensions are added to the toolkit in the form of extension plug-ins, which the toolkit will then be able to expose to the user. To provide an extension to Introduce, the extension provider must implement or extend the appropriate classes for the style of extension they wish to provide, and must fill out the extension XML configuration document. Once this extension is implemented and configured it can be placed in the `extensions` directory of Introduce. This directory has a common library (`lib`) area which enables it to avoid using custom class loaders for each extension.

4.6.1. SERVICE EXTENSIONS

```
<ns1:ExtensionDescription xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:ns1="gme://gov.nih.nci.cagrid.introduce/1/Extension" extensionType="SERVICE">
 <ns1:ServiceExtensionDescription displayName="Example Extension" name="service_example">
  <ns1:CreationPostProcessor>gov.nih.nci.cagrid.introduce.extension.example.ExampleCreationPostProcessor</ns1:CreationPostProcessor>
  <ns1:CreationUIDialog>gov.nih.nci.cagrid.introduce.extension.example.ExampleCreationDialog</ns1:CreationUIDialog>
  <ns1:CodegenPreProcessor>gov.nih.nci.cagrid.introduce.extension.example.ExampleCodegenPreProcessor</ns1:CodegenPreProcessor>
  <ns1:CodegenPostProcessor>gov.nih.nci.cagrid.introduce.extension.example.ExampleCodegenPostProcessor</ns1:CodegenPostProcessor>
  <ns1:ServiceModificationUIPanel>gov.nih.nci.cagrid.introduce.extension.example.ExampleServiceModificationPanel</ns1:ServiceModificationUIPanel>
 </ns1:ServiceExtensionDescription>
</ns1:ExtensionDescription>
```

Figure 10. Sample Service Extension Description.

A *service extension* is one which enables customization of the service creation and modification processes. These extensions can add required operations, service resources or resource properties, or security settings, for example. The service extension allows the user to provide custom code that will be executed at different times throughout the creation and modification processes of service development. A service extension consists of 5 main extension components that can be implemented and provided by the developer: CreationPostProcess, CreationUIDialog, CodegenPreProcess, CodegenPostProcess, and ServiceModificationUIPanel. Each of these extension components have a predefined class and interface that must be extended or implemented. Two of the service extension components, CreationUIDialog and ServiceModificationUIPanel, are graphical components provided to the Introduce GDE, and the other three are Introduce engine plug-ins.

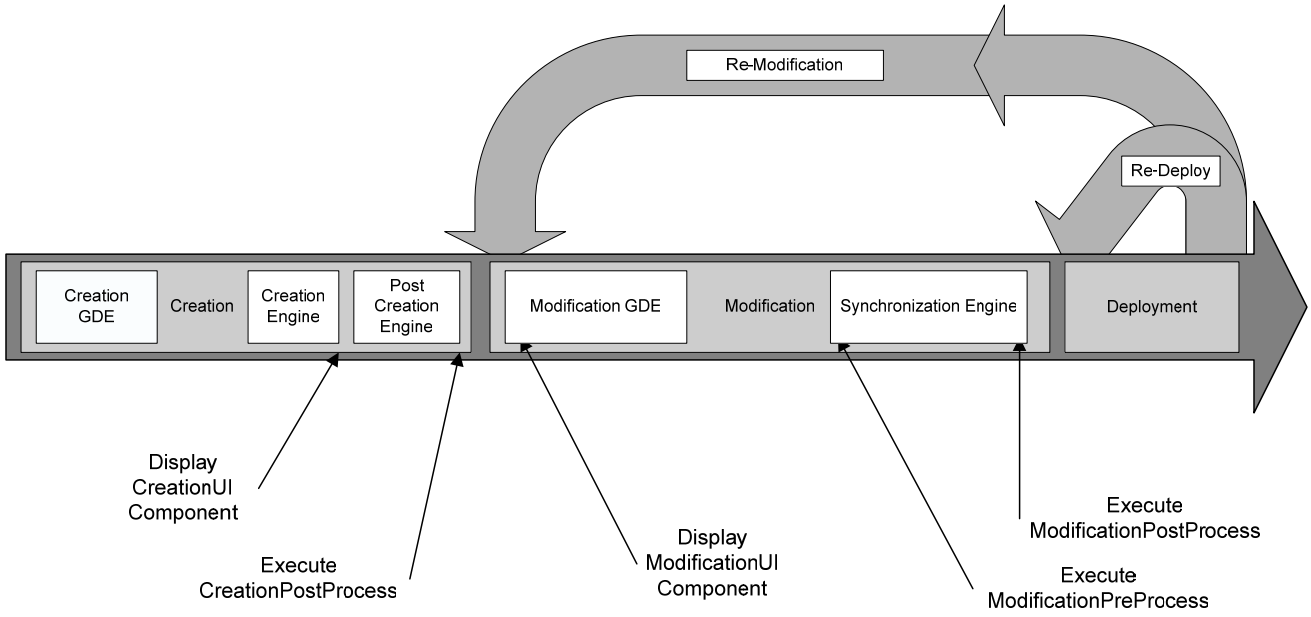


Figure 11. Execution of Introduce Extension Components.

Each service extension component is invoked at a specific point in the creation or modification steps (Figure 11). These different time points for each component execution are critical for making certain changes. For example, when the CreationUIDialog component for a particular extension is executed, the service has been created as a blank service and no modification or synchronization has been done on the service. At this point the CreationUIDialog might prompt the user for particular information about the creation processes. The CreationUI component would only be executed/displayed once for any given service and its non graphical component, the CreationPostProcess, will also only be ran one time after the service has been created and before it will ever be modified.

The modification components are ran every time a service is saved and synchronized. The graphical modification component is always available in the Introduce GDE during modification time. The two service modification engine components, ModificationPreProcess and ModificationPostProcess are executed respectively, before and after the synchronization process is executed. This enables ModificationPreProcess to do such things as modify the ServiceDesription, represented by the Introduce service description file, or the WSDL files of the services. The ModificationPostProcess, on the other hand, might move in required files, or populated stubbed methods, etc.

4.6.2. DATA TYPE DISCOVERY EXTENSIONS

```
<ns1:ExtensionDescription xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:ns1="gme://gov.nih.nci.cagrid.introduce/1/Extension" extensionType="DISCOVERY">
 <ns1:DiscoveryExtensionDescription displayName="Global Model Exchange" name="gme_discovery">
  <ns1:DiscoveryToolsPanelExtension>gov.nih.nci.cagrid.introduce.portal.discoverytools.gme.GMEViewer</ns1:DiscoveryToolsPanelExtension>
  <ns1:DiscoveryPanelExtension>gov.nih.nci.cagrid.introduce.portal.modification.discovery.gme.GMETypeSelectionComponent</ns1:DiscoveryPanelExtension>
  <ns1:Properties>
   <property key="GME_URL"
    value="http://cagrid01.bmi.ohio-state.edu:8080/wsrf/services/cagrid/GlobalModelExchange"/>
  </ns1:Properties>
 </ns1:DiscoveryExtensionDescription>
</ns1:ExtensionDescription>
```

Figure 12. Sample Discovery Tools Extension.

These extensions are Introduce GDE components that will be available at the service modification step. These components are intended to be able to provide custom data type discovery for the service developer. The custom data type discovery component must allow the developer to browse types and chose to use those types in the developed service. This means that the data type discovery extension will have to be able to copy the schemas which represent the data types down to the service's schema directory and produce a NamespaceType object for the namespace of each separate data type. This enables the grid service to utilize the schemas for describing the data types which are used in the WSDL messages traveling in and out of the created service.

In addition to default plug-ins, domain specific plug-ins can be installed in Introduce. For example, in the caBIG environment, a caDSR discovery plug-in is provided with Introduce. This plug-in allows service developers to locate and use data types registered in the cancer Data Standards Repository (caDSR), which is a curated repository of common data elements used in caBIG.

5. CONCLUSION

Grid computing has become the technology of choice that enables more effective use of distributed data, analytical, information, computational, and storage resources in coordinated, multi-institutional efforts and for large scale applications. The need for highly interoperable infrastructure is becoming more important in order to address the highly disparate and dynamic nature of organizations and groups within them. The Grid Services technology offers a viable platform to meet this need and facilitate the application of Grid computing in a wider range of application domains. Although a large number of middleware systems have been developed to support core Grid Services functionality, there are relatively few tools that provide high-level

functionality to support easy development of Grid Services. In this paper, we have proposed and presented a toolkit, called Introduce, that is designed to facilitate rapid development of *strongly-typed*, WSRF-compliant secure Grid Services and that supports the main steps of service development; service creation, service modification, and service deployment. In addition to ease-of-use and hiding the complexity of low-level Grid middleware infrastructure, Introduce encapsulates several novel features: 1) It supports development of strongly-typed services, in which input and output parameters of service methods are drawn from published, community-accepted data types. Strongly-typed services enables increased syntactic interoperability among services and are critical to programmatic access to services and correct handling of data returned from services. 2) It allows for service developers to easily leverage advanced features of WSRF and the Globus Toolkit 4 such as resource framework, multi-resource/multi-service, and compositional inheritance. 3) It provides a powerful extension framework for Introduce users to customize and extend the base Introduce functionality. Building on the notion of plug-ins, this framework goes beyond simple extensibility via service templates and enables integration of both custom code/template generation logic and custom graphical components as complex plug-ins.

The design of Introduce is motivated mainly by the caBIG effort and its Grid infrastructure, called caGrid, designed to support implementation, execution, and management of services and applications in the caBIG environment. Introduce has been successfully employed in the development of services within the caBIG caGrid infrastructure as well as data and analytical services that make use of the caGrid functionality. We believe that Grid service development tools such as Introduce and other related technologies will help bring the technology of Grid Computing to the forefront of distributed infrastructure and greatly increase the adoptability of grid service technologies in areas from scientific research to business computing.

In the future we plan to provide even higher level of service interoperability at the semantic level. We plan to investigate the notion of not only designing services from common public data types but also possible from publicly available modeled operation signatures or even service interfaces. We anticipate that increasing the availability of these types of information and providing tools which can aid in the creation of grid services from them will enable a higher level of semantic interoperability, which will lead to API harmonization across domains.

References

[1] B. Allcock, J. Bester, J. Bresnahan, A. Chervenak, I. Foster, C. Kesselman, S. Meder, V. Nefedova, D. Quesnal, and T. S., "Data Management and Transfer in High Performance Computational Grid Environments," *Parallel Computing Journal*, vol. 28, pp. 749-771, 2002.

[2] W. E. Allcock, I. Foster, and R. Madduri, "Reliable Data Transport: A Critical Service for the Grid.," in *Proceedings of Building Service Based Grids Workshop, Global Grid Forum 11*. Honolulu, Hawaii, USA, 2004.

[3] G. Allen, T. Dramlitsch, I. Foster, T. Goodale, N. Karonis, M. Ripeanu, E. Seidel, and B. Toonen, "Cactus-G Toolkit: Supporting Efficient Execution in Heterogeneous Distributed Computing Environments," in *Proceedings of the 4th Globus Retreat*. Pittsburg, PA, 2000.

[4] H. Andrade, T. Kurc, A. Sussman, and J. Saltz, "Active Proxy-G: Optimizing the Query Execution Process in the Grid," in *Proceedings of the ACM/IEEE Supercomputing Conference (SC2002)*. Baltimore, MD: ACM Press/IEEE Computer Society Press, 2002.

[5] J. Annis, Y. Zhao, J. Voeckler, M. Wilde, S. Kent, and I. Foster, "Applying Chimera Virtual Data Concepts to Cluster Finding in the Sloan Sky Survey," in *Proceedings of the ACM/IEEE Supercomputing Conference (SC2002)*. Baltimore, MD: ACM Press/IEEE Computer Society Press, 2002.

[6] M. P. Atkinson and et.al., "Grid Database Access and Integration: Requirements and Functionalities," Technical Document, Global Grid Forum. http://www.cs.man.ac.uk/grid-db/documents.html, 2002.

[7] F. Berman, H. Casanova, J. Dongarra, I. Foster, C. Kesselman, J. Saltz, and R. Wolski, "Retooling Middleware for Grid Computing," *NPACI & SDSC enVision*, vol. 18, 2002.

[8] M. Beynon, T. Kurc, A. Sussman, and J. Saltz, "Design of a Framework for Data-Intensive Wide-Area Applications," in *Proceedings of the 2000 Heterogeneous Computing Workshop (HCW2000)*. Cancun, Mexico, 2000.

[9] H. Casanova, O. Graziano, F. Berman, and R. Wolski, "The AppLeS Parameter Sweep Template: User-Level Middleware for the Grid," in *Proceedings of the ACM/IEEE Supercomputing Conference (SC2000)*: ACM Press/IEEE Computer Society Press, 2000.

[10] A. Chervenak, E. Deelman, I. Foster, L. Guy, W. Hoschek, A. Iamnitchi, C. Kesselman, P. Kunst, M. Ripeanu, B. Schwartzkopf, H. Stockinger, and B. Tierney, "Giggle: A Framework for Constructing Scalable Replica Location Services," in *Proceedings of the ACM/IEEE Supercomputing Conference (SC2002)*: ACM Press/IEEE Computer Computer Society Press, 2002, pp. 1-17.

[11] A. Chervenak, E. Deelman, C. Kesselman, B. Allcock, I. Foster, V. Nefedova, J. Lee, A. Sim, A. Shoshahi, B. Drach, D. Williams, and D. Middleton, "High-performance remote access to climate simulation data: a challenge problem for data grid technologies," *Parallel Computing*, vol. 29, pp. 1335-1356, 2003.

[12] A. Chervenak, I. Foster, C. Kesselman, C. Salisbury, and S. Tuecke, "The Data Grid: Towards an Architecture for the Distributed Management and Analysis of Large Scientific Datasets," *Journal of Network and Computer Applications*, vol. 23, pp. 187-200, 2000.

[13] E. Deelman, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Vahi, K. Blackburn, A. Lazzarini, A. Arbree, R. Cavanaugh, and S. Koranda, "Mapping Abstract Complex Workflows onto Grid Environments," *Journal of Grid Computing*, vol. 1, pp. 25-39, 2003.

[14] E. Deelman, G. Singh, M. P. Atkinson, A. Chervenak, N. P. Chue Hong, C. Kesselman, S. Patil, L. Pearlman, and M. Su, "Grid-Based Metadata Services," in *Proceedings of the 16th*

International Conference on Scientific and Statistical Database Management (SSDBM '04), 2004.

[15] I. Foster and C. Kesselman, "Globus: A Metacomputing Infrastructure Toolkit.," *International Journal of High Performance Computing Applications*, vol. 11, pp. 115--128, 1997.

[16] I. Foster, J. Voeckler, M. Wilde, and Y. Zhao, "Chimera: A Virtual Data System for Representing, Querying, and Automating Data Derivation," in *Proceedings of the 14th Conference on Scientific and Statistical Database Management (SSDBM '02)*, 2002.

[17] J. Frey, T. Tannenbaum, M. Livny, I. Foster, and S. Tuecke, "Condor-G: A Computational Management Agent for Multi-institutional Grids," in *Proceedings of the Tenth International Symposium on High Performance Distributed Computing (HPDC-10)*: IEEE Press, 2001.

[18] N. Furmento, W. Lee, A. Mayer, S. Newhouse, and J. Darlington, "ICENI: An Open Grid Service Architecture Implemented with JINI," in *Proceedings of the ACM/IEEE Supercomputing Conference (SC2002)*. Baltimore, MD: ACM Press/IEEE Computer Society Press, 2002.

[19] A. S. Grimshaw and W. Wulf, "The Legion: Vision of a Worldwide Virtual Computer," *Communications of the ACM*, vol. 40, pp. 39--45, 1997.

[20] S. Hastings, S. Langella, S. Oster, and J. Saltz, "Distributed Data Management and Integration: The Mobius Project," *Proceedings of the Global Grid Forum 11 (GGF11) Semantic Grid Applications Workshop, Honolulu, Hawaii, USA.*, pp. 20-38, 2004.

[21] S. Langella, S. Oster, S. Hastings, F. Siebenlist, T. Kurc, and J. Saltz, "Dorian: Grid Service Infrastructure for Identity Management and Federation," presented at The 19th IEEE Symposium on Computer-Based Medical Systems, Special Track: Grids for Biomedical Informatics, Salt Lake City, Utah., 2006.

[22] R. Oldfield and D. Kotz, "Armada: A Parallel File System for Computational Grid," in *Proceedings of the IEEE International Symposium on Cluster Computing and the Grid (CCGrid2001)*. Brisbane, Australia: IEEE Computer Society Press, 2001.

[23] M. Sato, H. Nakada, S. Sekiguchi, S. Matsuoka, U. Nagashima, and H. Takagi, "Ninf: A Network based Information Library for a Global World-Wide Computing Infrastructure," in *Proceedings of the Conference on High Performance Computing and Networking (HPCN '97) (LNCS-1225)*, 1997, pp. 491-502.

[24] G. Singh, S. Bharathi, A. Chervenak, E. Deelman, C. Kesselman, M. Mahohar, S. Pail, and L. Pearlman, "A Metadata Catalog Service for Data Intensive Applications," in *Proceedings of the ACM/IEEE Supercomputing Conference (SC2003)*, 2003.

[25] G. Singh, E. Deelman, G. Mehta, K. Vahi, M. Su, B. Berriman, J. Good, J. Jacob, D. Katz, A. Lazzarini, K. Blackburn, and S. Koranda, "The Pegasus Portal: Web Based Grid Computing," in *Proceedings of the 20th Annual ACM Symposium on Applied Computing*. Santa Fe, New Mexico, 2005.

[26] J. Smith, A. Gounaris, P. Watson, N. W. Paton, A. A. Fernandes, and R. Sakellariou, "Distributed Query Processing on the Grid.," presented at Proceedings of the Third Workshop on Grid Computing (GRID2002), Baltimore, MD, 2003.

[27] D. Thain, J. Basney, S. Son, and M. Livny, "Kangaroo Approach to Data Movement on the Grid," in *Proceedings of the Tenth IEEE Symposium on High Performance Distributed Computing (HPDC-10)*, 2001.

[28] L. Weng, G. Agrawal, U. Catalyurek, T. Kurc, S. Narayanan, and J. Saltz, "An Approach for Automatic Data Virtualization," in *Proceedings of the 13th IEEE International Symposium on High-Performance Distributed Computing (HPDC-13)*. Honolulu, Hawaii, 2004, pp. 24-33.

[29] I. Foster, C. Kesselman, J. M. Nick, and S. Tuecke, "The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration," Open Grid Service Infrastructure Working Group Technical Report, Global Grid Forum. http://www.globus.org/alliance/publications/papers/ogsa.pdf 2002.
[30] I. Foster, C. Kesselman, and S. Tuecke, "The Anatomy of the Grid: Enabling Scalable Virtual Organizations.," *International Journal of Supercomputer Applications*, vol. 15, pp. 200-222, 2001.
[31] E. Cerami, *Web Services Essentials*: O'Reilly & Associates Inc., 2002.
[32] S. Graham, S. Simeonov, T. Boubez, D. Davis, G. Daniels, Y. Nakamura, and R. Neyama, *Building Web Services with Java: Making Sense of XML, SOAP, WSDL, and UDDI*: SAMS Publishing, 2002.
[33] K. Czajkowski, D. F. Ferguson, I. Foster, J. Frey, S. Graham, I. Sedukhin, D. Snelling, S. Tuecke, and W. Vambenepe, "The WS-Resource Framework version 1.0," vol. 2004, 2004.
[34] J. Saltz, S. Oster, S. Hastings, T. Kurc, W. Sanchez, M. Kher, A. Manisundaram, K. Shanbhag, and P. Covitz, "caGrid: Design and Implementation of the Core Architecture of the Cancer Biomedical Informatics Grid," *Bioinformatics. (in press).* 2006.
[35] S. Langella, S. Hastings, S. Oster, T. Kurc, U. Catalyurek, and J. Saltz, "A Distributed Data Management Middleware for Data-Driven Application Systems," in *Proceedings of the 2004 IEEE International Conference on Cluster Computing (Cluster 2004)*, 2004.
[36] K. Bhatia, S. Chandra, and K. Mueller, "GAMA: Grid Account Management Architecture," San Diego Supercomputer Center (SDSC), UCSD Technical Report. #TR-2005-3, 2005.
[37] I. Foster, C. Kesselman, S. Tuecke, V. Volmer, V. Welch, R. Butler, and D. Engert, "A National Scale Authentication Infrastructure," *IEEE Computer*, vol. 33, pp. 60-66, 2000.
[38] V. Welch, F. Siebenlist, I. Foster, J. Bresnahan, K. Czajkowski, J. Gawor, C. Kesselman, S. Meder, L. Pearlman, and S. Tuecke, "Security for Grid Services," presented at 12th International Symposium on High Performance Distributed Computing (HPDC-12), 2003.
[39] H. Morohoshi and R. Huang, "A User-friendly Platform for Developing Grid Services over Globus Toolkit 3," presented at The 2005 11th International Conference on Parallel and Distributed Systems (ICPADS'05), 2005.
[40] S. Mizuta and R. Huang, "Automation of Grid Service Code Generation with AndroMDA for GT3," presented at The 19th International Conference on Advanced Information Networking and Applications (AINA'05), 2005.
[41] G. von Laszewski, I. Foster, J. Gawor, and P. Lane, "A Java Commodity Grid Kit," *Concurrency and Computation: Practice and Experience*, vol. 13, pp. 643-662, 2001.
[42] G. von Laszewski, I. Foster, J. Gawor, W. Smith, and S. Tuecke, "CoG Kits: A Bridge Between Commodity Distributed Computing and High Performance Grids," presented at ACM Java Grande 2000 Conference, 2000.
[43] R. Buyya and S. Venugopal, "The Gridbus Toolkit for Service Oriented Grid and Utility Computing: An Overview and Status Report," presented at the First IEEE International Workshop on Grid Economics and Business Models (GECON 2004), New Jersey, USA, 2004.
[44] M. Humphrey and G. Wasson, "Architectural Foundations of WSRF.NET," *International Journal of Web Services Research*, vol. 2, pp. 83-97, 2005.
[45] M. Smith, T. Friese, and B. Freisleben, "Model Driven Development of Service Oriented Grid Applications," presented at Advanced International Conference on Telecommunications and

International Conference on Internet and Web Applications and Services (AICT-ICIW '06), 2006.

Appendix.

1. Examples of the various files and directories managed by Introduce for a service.

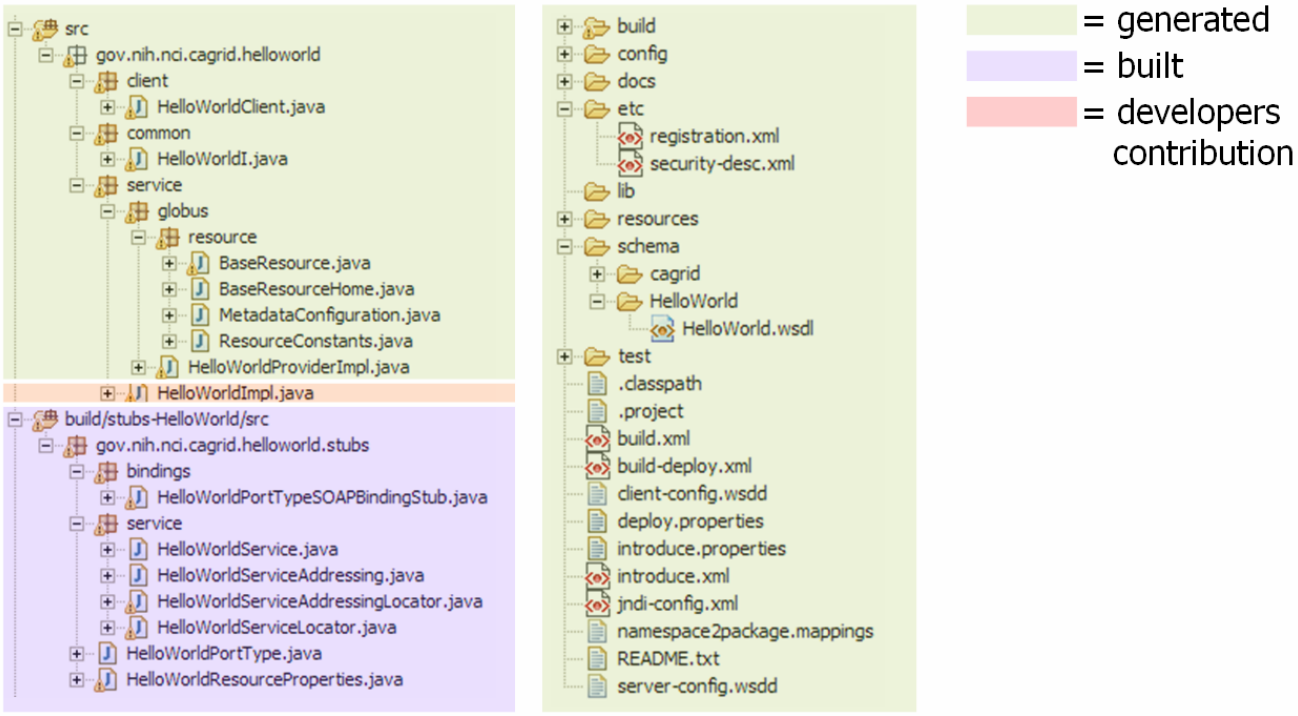


Figure A1. Basic service layout of an example service created by Introduce .

Figure A1 shows a basic service layout as created by the Introduce Service Creator component. It shows which pieces of this example service are generated by the code generation tools, which pieces are built by the underlying Globus/Axis tools, and which pieces are to be implemented by the service developer.

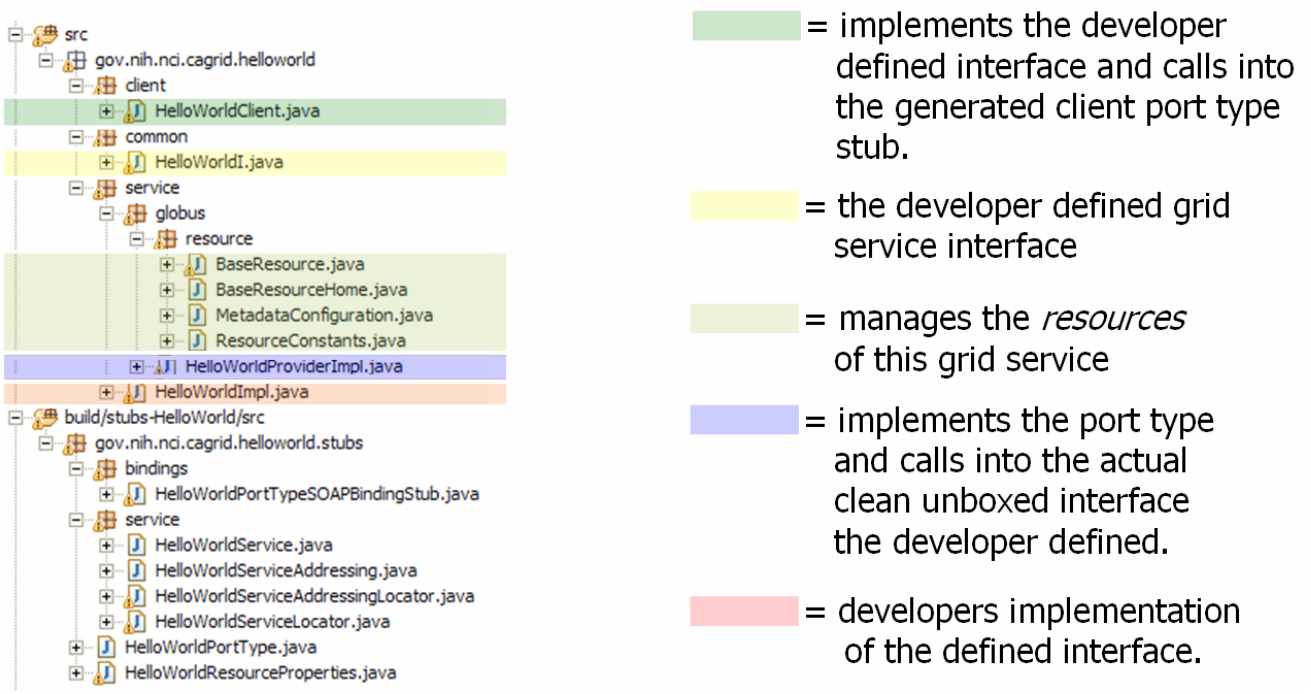


Figure A2. Source files used in an example service created by Introduce.

Figure A2 describes what some particularly critical source files are generated for in an example service created by the Introduce engine. When a WSDL file is parsed by the Axis engine a PortType interface is created which is the Java representation of the API of the grid service. The Axis generated PortType interface must then be implemented on the service to provide the services implementation. In order to enable the service developer to implement a cleaner, non-documument literal interface, Introduce will automatically create the implementation of this PortType interface (*HelloWorldProviderImpl*). The Introduced generated implementation of this interface will *unbox* the document literal calls ot the service and pass them on to the unboxed/clean interface (*HelloWorldI*) which the user defined. Introduce will generate a stubbed implementation of this interface (*HelloWorldImpl*) which the user will be responsible for implementing. This class will maintain the services implementation of the methods. This enables the service designer to be shielded from the details of the Axis document literal grid service interface and enable them to implement an interface which is as then originally described. The figure also illustrates the example service's resource and resource home, which are generated to manage the service resource and the service resource properties, respectively.

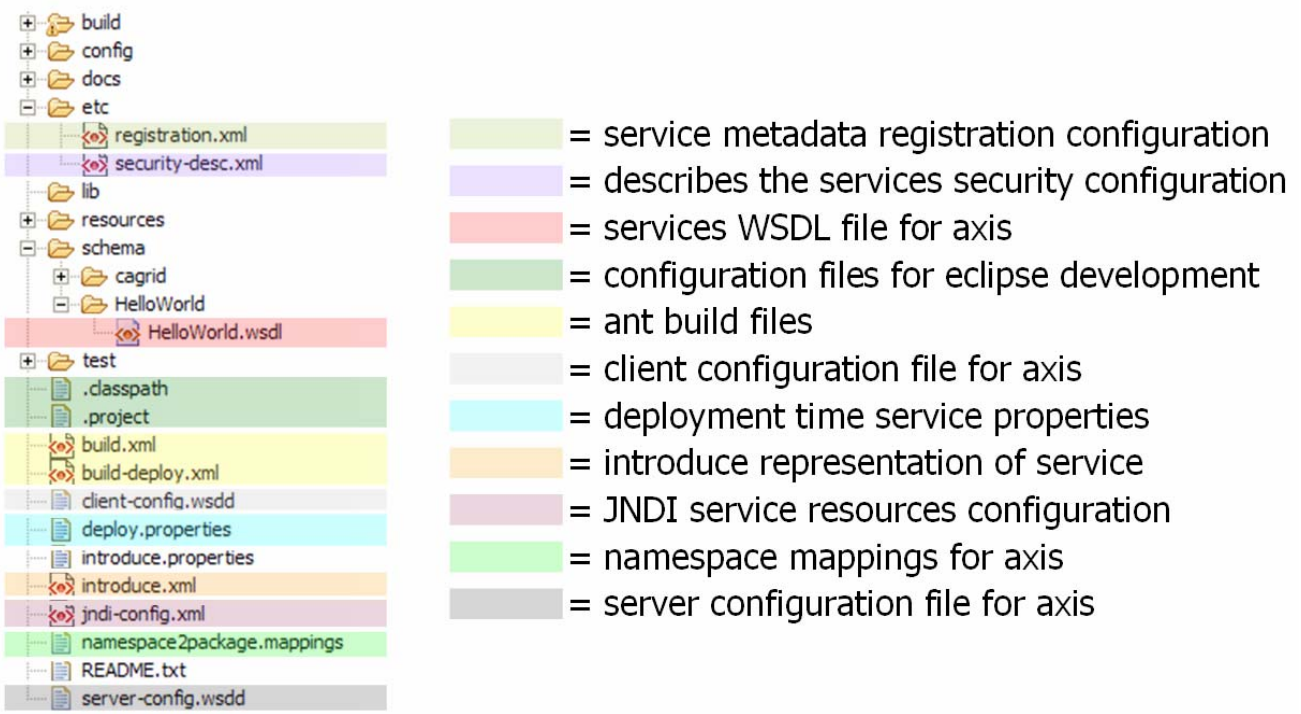


Figure A3 Example common/configuration files on an example service created by Introduce.

Figure A3 shows the use of the different common files of an example Introduce created service. It shows the files used for configuring registration and security for the grid service, as well as those used by Introduce for synchronization, and those used for build and deployment. This example service created by Introduce also contains Eclipse project files so that the service can easily be edited using the Eclipse platform (www.eclipse.org).