J Grid Computing (2007) 5:407–427
DOI 10.1007/s10723-007-9074-8

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

Received: 7 June 2006 / Accepted: 15 February 2007 / Published online: 9 March 2007

Abstract Service-oriented architectures 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 architectures. Most of these efforts, however, have focused on 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 would allow a Grid service developer to leverage Grid technologies without needing to know low-level details, are becoming increasingly important for wider application of the Grid. In this paper, we describe an open-source, extensible toolkit, called Introduce, that supports easy development and deployment of Web Services Resource Framework (WSRF) compliant services. Introduce is designed 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 services. In strongly typed services, a service produces and consumes data types that are well-defined and published in the Grid. This enables data-level syntactic interoperability so that clients and services can access and consume data elements programmatically and correctly. 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 in the commercial sector as well as in academic, medical, and government research.

Keywords Grid · Grid computing · Web services · Grid service · Globus · WSDL

S. Hastings (✉) · S. Oster · S. Langella · D. Ervin · T. Kurc · J. Saltz
Department of Biomedical Informatics,
The Ohio State University,
Columbus, OH, USA
e-mail: hastings@bmi.osu.edu

S. Oster
e-mail: oster@bmi.osu.edu

S. Langella
e-mail: langella@bmi.osu.edu

D. Ervin
e-mail: ervin@bmi.osu.edu

T. Kurc
e-mail: kurc@bmi.osu.edu

J. Saltz
e-mail: jsaltz@bmi.osu.edu

1 Introduction

Grid computing is quickly becoming the new means for creating distributed infrastructures and virtual organizations for multi-institutional research and enterprise applications. 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 as well as computational and storage resources. A vast array of middleware systems and toolkits has been developed to support applications in the Grid. These include middleware and tools for service deployment and remote service invocation [1], security [2–4], resource monitoring and scheduling [5–8], high-speed data transfer [9–11], metadata and replica management [12–16], component based application composition [17–19], and workflow management [20–22]. Earlier efforts 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 managed autonomously, and programmatic access to resources is desired. In recent years, a services oriented view of the Grid has emerged to address interoperability concerns. The Open Grid Services Architecture (OGSA) [23, 24] has been developed through standardization efforts coordinated mainly by the Global Grid Forum (http://www.ggf.org), which includes both academic researchers and corporate IT leaders, and OASIS (http://www.oasis-open.org). OGSA specifies a Grid Services framework and Grid Services standards. The Grid Services framework builds on and extends Web Services [25, 26] for scientific applications. It defines mechanisms for such additional features as stateful services, service notification, and management of service/resource lifetime. A more recent effort, the Web Services Resource Framework (WSRF) standardization [27], has paved the way for closer interoperability and unification between stateful Grid services and Web services.

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 deployment and invocation of Grid services, and provides runtime support for common services such as an Index service for service registration and discovery and the Globus Security Infrastructure (GSI) for security. These components enable service providers to stand up Grid services, advertise them, discover them, and secure them. The OGSA, WSRF, and GT provide a foundation upon which domain specific secure services, tools, and applications can be developed. However, additional tools that will make it easier to use Grid technologies in service and application development and deployment are needed for wider adoption of the Grid in application domains.

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 goals of our work are twofold. The first is to reduce the development time and knowledge required to implement and stand up Grid Services using the GT. Developing services with the Globus Toolkit requires a good knowledge of Grid Services technologies and of the details of the toolkit. Introduce hides from service developers the complexities of low-level tools and processes for service development and deployment. The second goal is to enable greater levels of interoperability in the Grid environment. To this end, we implement support in Introduce for development of *strongly typed* services. A strongly typed service is one that consumes and produces data types that are well-defined and published in the 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. We believe that the availability of tools like Introduce will greatly impact the efficacy of the Grid in fields where Grid style architectures are a great change to the everyday development and management of data and analytical services.

The current implementation of Introduce uses the GT as the underlying core Grid infrastructure and the Global Model Exchange (GME) service of the Mobius framework [16, 28, 29] to support strongly typed Grid service development. The salient features of Introduce can be summarized as follows:

- It provides a graphical user interface (GUI) and high-level functions 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" document literal 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 GUI. Developers can also write their own service description documents, which the Introduce toolkit can use to create and modify a service programmatically.
- It enables development of strongly typed services. Using 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 the 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 applications to interact with the service.

Introduce is available with the caGrid version 0.5 and 1.0 distributions (https://cabig.nci.nih.gov/workspaces/Architecture/caGrid); caGrid is the Grid architecture and middleware infrastructure of the NCI-funded cancer Biomedical Informatics Grid (caBIG™; https://cabig.nci.nih.gov) program [30]. Originally implemented as a tool to assist analytical service developers in caBIG, Introduce has evolved into a more generic Grid service development and deployment toolkit. The current version of Introduce has been used as the core caBIG service development toolkit in caGrid version 1.0, which was released in December 2006, to build various types of services ranging from data services to core security infrastructure services to biomedical image analysis services. Introduce was also accepted as an Incubator project for the GT in November 2006; it is planned to be a subproject of the GT after the incubation period.

The rest of this paper is organized as follows. We present the motivation behind Introduce and the notion of strongly typed service in greater detail in Section 2. Section 3 provides a review of related work in toolkits for Grid application development. The main components of the Introduce toolkit are described in Section 4. Additional features of Introduce and its extension framework are presented in Sections 5 and 6. We conclude in Section 7.

2 Motivation

The need for the Introduce toolkit primarily arose in 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 Grid framework. In this section, using caBIG as an example, we discuss the motivations for the two main goals of the Introduce effort; easy development of secure Grid services and support for strongly typed service development.

2.1 Easy Development and Deployment of Secure Grid Services

The principle idea in caBIG is to leverage combined strengths of individual cancer centers and research labs by creating common applications, standards, common data and analytical resources, and software infrastructure to link these applications and resources. In this effort, a core Grid infrastructure, called *caGrid*, is being developed to facilitate sharing of data and analytical resources, and to support development of Grid-enabled applications that make use of these resources. The infrastructure is designed as a service-oriented system, building on the WSRF standards.[1] Being the most complete and widely deployed reference implementation of the WSRF standard, the Globus Toolkit (GT) is employed as the Grid

[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 developed using the WSRF standards and Globus Toolkit 4.

middleware backbone and runtime environment in caGrid.

caBIG is envisioned to span hundreds of institutions and possibly thousands of investigator and research laboratories. These institutions may host software systems of different complexity and maturity. They may 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 and directories that are required for successful compilation and deployment of services. The Introduce toolkit helps a service developer by coordinating the various steps of the service development and deployment via the tools provided by the GT and by managing 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. The caGrid infrastructure provides higher level services such as Dorian [4] and the Grid Trust Service (GTS) for managing and enforcing security. Introduce allows a service developer to configure the service as a *secure* service. When configured, the service can interact with Grid-wide and local authentication and authorization services to enforce controlled access to its methods.

2.2 Strongly Typed Service Development

Interoperability is critical in environments where it is desirable to access heterogeneous resources programmatically and the resources are developed and managed by different groups independently. In caBIG, for example, there can be multiple data sources that maintain databases of Gene expression data at different institutions; similarly, there can be multiple analytical resources providing microarray analysis methods. To make the sharing of and remote access to such resources easy and efficient in caBIG, applications should be able to access the resources through common interface syntax and protocols, and interact with their data structures programmatically.

The Introduce toolkit aims to support and enhance interoperability in two main categories; architecture-level and information-level interoperability. The *architecture-level interoperability* requires use of common data and control information exchange protocols and common interface syntax for programmatic access to the functionality of a Grid resource. The OGSA and WSRF enable architecture-level interoperability by defining standards for data and information exchange protocols, service creation and management, and service invocation. A Web or Grid service exposes its structure and interface definitions in WSDL so that clients can invoke the service's methods and interact with the service remotely. Introduce leverages WSRF to support architecture-level interoperability. The *information-level interoperability*, on the other hand, is concerned with programmatic access to information produced by a Grid resource and the correct interpretation and consumption of the information. It is especially important when information is exchanged between disjoint groups of data providers and consumers. The information-level interoperability requires the use of controlled vocabularies, common data types, and common information models to represent the information. In the context of information-level interoperability, we define the notion of *strongly typed service* as the class of services in which the input and output data types of service methods are well-defined and registered in the Grid environment.

We developed the concept of a strongly typed service first in the Mobius project [16, 29] in the context of XML virtualization of data sources. In Mobius, the structure of a XML document or a data element provided by a service is required to conform to a XML schema registered in the environment. In caBIG, information-level interoperability is addressed through coordinated management of metadata and curated objects, bound to underlying semantic concepts. Objects are exchanged between two Grid end

points (i.e., between services or between clients and services) in the form of XML documents. The structure of an object is represented by a registered and published XML schema. That is, a strongly typed caBIG service has service methods, whose input and output data types conform to registered XML schemas.[2] The Introduce toolkit makes use of XML schemas to enable strongly typed service development and define data types.

The support for strongly typed services requires the availability of data type repository services, which maintain the definition and structure of Grid supplied data types. Data type repositories are used by service developers to locate data types which suite their needs and use them as method input and output parameters. Having Grid-wide accessible data type repository services can promote the usage of common data types across the Grid. Introduce leverages the Global Model Exchange (GME) service of the Mobius framework as a data type repository. The GME provides tools and services for coordinated management of XML schemas. A XML 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. 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 domains which publish semantic ontologies, semantic interoperability can also be facilitated through the use of ontology aware data type discovery extensions of Introduce.

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

3 Related Work

The Globus Service Build Tools Project (http://gsbt.sourceforge.net/) consists of GT4IDE and globus-build-services components. The GT4IDE component is an Eclipse (http://www.eclipse.org) plug-in designed to aid in the development of GT 4.0 compatible services. 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 leverages tools provided by the Eclipse project such as the Java Emitter Templates (JET), but is a stand-alone application, the GT4IDE plug-in can only be used in the Eclipse development environment. Introduce has several additional features, which are not part of the GT4IDE implementation. These features include support for strongly typed services, hiding of document literal bindings behind user defined service interfaces, extensibility, and ability to leverage advanced features of GT 4.0 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 Sun Studio Creator, JAXR, and JAX-WS 2.0 are Java based toolkits for development of Web applications and Web Services (http://java.sun.com/webservices/index.jsp) and provide an integrated environment to support development life cycle. Introduce is a toolkit targeted at WSRF compliant services. 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 GT 3.0 platform. Unlike Introduce, which provides access to service development and deployment functions from a graphical user interface, the IBM Toolbox 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.

Smith et. al. and Friese et. al. [31, 32] lay 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 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 security, registration, invocation, or composition processes. 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.

gEclipse is a recent project in its definition stages (http://www.geclipse.eu/). It will develop a framework as plug-in extensions to the Eclipse infrastructure to both integrate various existing tools in a unified environment and build new tools for easy development of Grid applications using Java, C/C++, and other programming languages. Our work differs from gEclipse in that our effort is focused on development of WSRF compliant services, strongly typed service development, and secure service configuration and deployment.

Morohoshi and Huang [33] 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 toolkit GUI 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), and deploy or undeploy the service. Mizuta and Huang [34] 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. Their approach allows a service developer to use existing UML-based modeling systems to design Grid services. The cartridge developed for the AndroMDA implements the necessary functions and extensions so that all the files, directories, source codes, and service data elements are generated for the service under development. One limitation of their work is their tool can be used only in the AndroMDA framework. Introduce has several features not available in these toolkits. These features include (1) a more comprehensive extension framework (see Section 6), 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 so that a service can be designed using UML models.

The Java CoG kits [35, 36] offer a rich set of client side capabilities to develop clients and access Grid functions and services. Limited support is provided for server side capabilities and service development. The goal of the WSRF .NET [37] project is 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.

P-GRADE [38] and ASKALON [39] are examples of toolkits that provide support for development of high-performance applications on cluster platforms and in the Grid. ASKALON provides a suite of tools in a service-oriented architecture for performance measurement, experiment management, performance bottleneck detection, and performance prediction. P-GRADE implements a graphical interface and runtime support to develop and execute PVM and MPI parallel programs in the Grid. It supports both interactive and job mode execution of program and uses scheduling systems like Condor-G [7] for job scheduling and execution. The graphical interface allows a developer to define components, component processes, and communication patterns between com-

ponent processes. GEMLCA [40] is a suite of tools and services that facilitate deployment of a legacy program as a Grid service. It employs a non-intrusive approach in that deployment can be carried out without making code modifications to the legacy application; the owner of the application provides a description of the input and output parameters of the application, where the application is located, and the execution environment of the application in a Legacy Code Interface Description file. The GEMLCA components and services allow a user to register and deploy a legacy application in the framework, invoke it remotely through Grid service interfaces, and get the status and output of the execution. FORTE HPC (http://developers.sun.com/sunstudio/products/previous/fortran/index.html) provides an integrated environment for development, debugging, and execution of high-performance applications written in Fortan and C. Introduce is complementary to these toolkits in that it provides tools for development of WSRF compliant services. P-GRADE, ASKALON, FORTE HPC, and GEMLCA as well as tools for high-performance bulk data transfer [10, 11], job scheduling and monitoring [5, 7], could be used in the Introduce framework to support services that involve parallel application execution, legacy programs, job scheduling, and large data transfer.

The Grid Interoperation Now (GIN) community group of the Open Grid Forum (OGF) (https://forge.gridforum.org/sf/projects/gin) is leading efforts to facilitate operational and infrastructure level interoperability between different production Grids. The main areas focused on by the group include interoperability between authorization and authentication services, data management and movement, job description and submission, and information services. Introduce promotes the development and use of strongly typed services through the use of well-defined and published data types in WSRF compliant services. This aspect of Introduce can be viewed as one of the enabling technologies for interoperability at the level of data structures produced and consumed by services in different Grids. The User Program Development Tools research group (http://www.ggf.org/7_APM/UPDT.htm) has investigated tools to support development cycle of Grid-enabled applications with a focus on tools for debugging and tuning applications. Introduce is a program development tool, but focuses on support for exposing data and analytical resources as WSRF compliant services. An application developer could use performance tuning and debugging tools along with Introduce to optimize the performance of their service implementation.

4 Introduce Toolkit

The Introduce toolkit is designed to support the three main steps of service development (see Fig. 1): (a) *Creation*: 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 service configuration properties, Introduce will create the basic service implementation, to which the developer can add application-specific methods and security options through the service modification steps. (b) *Modification*: The modification step allows the developer to add, remove, and modify service methods, properties,

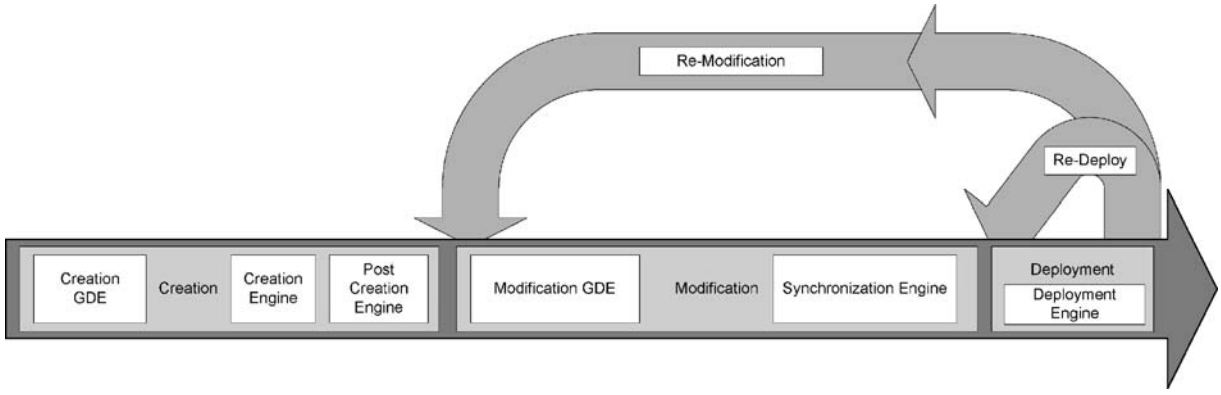

Fig. 1 Overall service development process

resources, service contexts, and service/method level security configuration. In this step, the developer creates strongly typed service interfaces using well-defined schemas, which are registered in a system like the Mobius GME as type definitions of the input and output parameters of the service methods. (c) *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. In this section, we describe the GDE and the core components of the Introduce engine. Section 5 presents the additional features of the engine, including resource framework, support for compositional inheritance, and support for multi-service/multi-resource implementations. Introduce is an extensible toolkit. The extension framework, which allows for integration of custom service types and discovery of custom data types, is described in Section 6.

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, accessible locally or remotely, that the toolkit can pull common data types for strongly typed services. The toolkit is designed as a modular system, in which the specific implementations of external components can be replaced.

The current implementation of Introduce leverages the Globus Toolkit (GT), Apache Axis (http://ws.apache.org/axis/), and Java Naming and Directory Interface (JNDI; http://java.sun.com/products/jndi/). Apache Axis is an open source middleware providing the actual Web Services layer, i.e., SOAP based Web Service protocol 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. Introduce uses the Mobius GME service [16] 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.

4.2 Introduce Graphical Development Environment

The Introduce Graphical Development Environment (GDE) is the graphical service authoring tool which can be used to create, modify, and deploy a Grid service (see Fig. 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. These plug-ins enable custom extensions to Introduce that can be used by service developers.

The interface contains several screens and options for the service developer. The service creation screens enable 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 service modification screens allow the service developer to customize the service by adding, removing, and modifying service methods and the input and output parameters of the methods. The developer can also configure the service to enforce authentication and authorization via Grid security infrastructure. Using the data types registration and discovery screens, the developer can obtain the data types that he/she wants to use for the service parameters and return types from any data type discovery plug-in. 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 deployment component of the GDE allows the service developer to deploy the implemented Grid service 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.

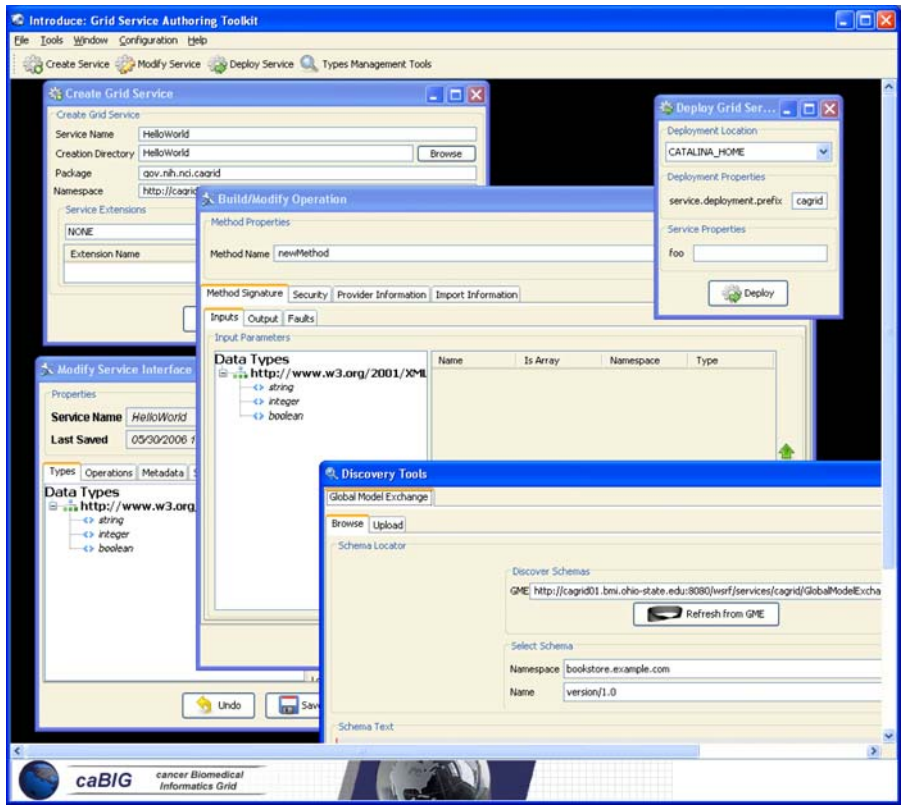

Fig. 2 The Introduce Graphical Development Environment (GDE)

4.3 Introduce Engine

The runtime support to enable service creation, modification, and deployment of Grid services is provided by the Introduce engine. The engine is the core piece that tracks changes in a service being implemented, modifies the service to reflect these changes, and rebuilds the service to verify that it is ready to be edited or deployed. In this section, we describe the main components of this engine.

4.3.1 Service Creator

The service creator is composed of 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.

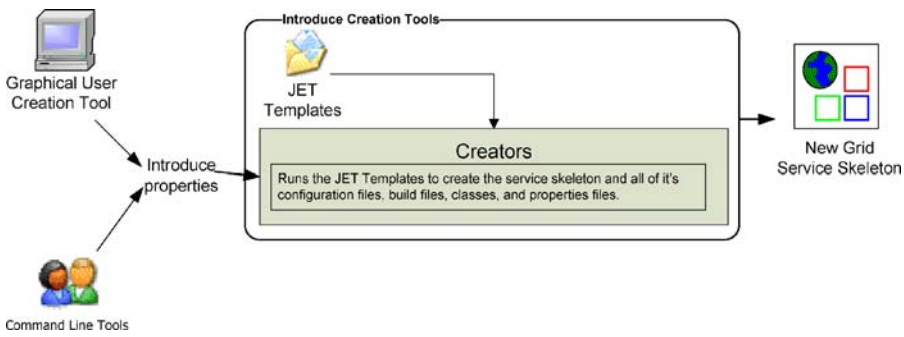

Fig. 3 Service creation tools and use of JET templates for service creation

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 Fig. 3). 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, Web Service Deployment Descriptor (WSDD), and security configuration. The basic service created by Introduce is the skeleton of the service application developer wants to develop and it does not include such additions as multi-service/multi-source properties and compositional inheritance properties. Such additional configuration and modifications can be performed in the service modification step. The basic service consists of the set of method interfaces defined by the service developer and has the core set of files and processed needed to compile the service and deploy by the lower level Grid middleware:

- ANT processes (http://ant.apache.org) for build, deploy, and test operations,

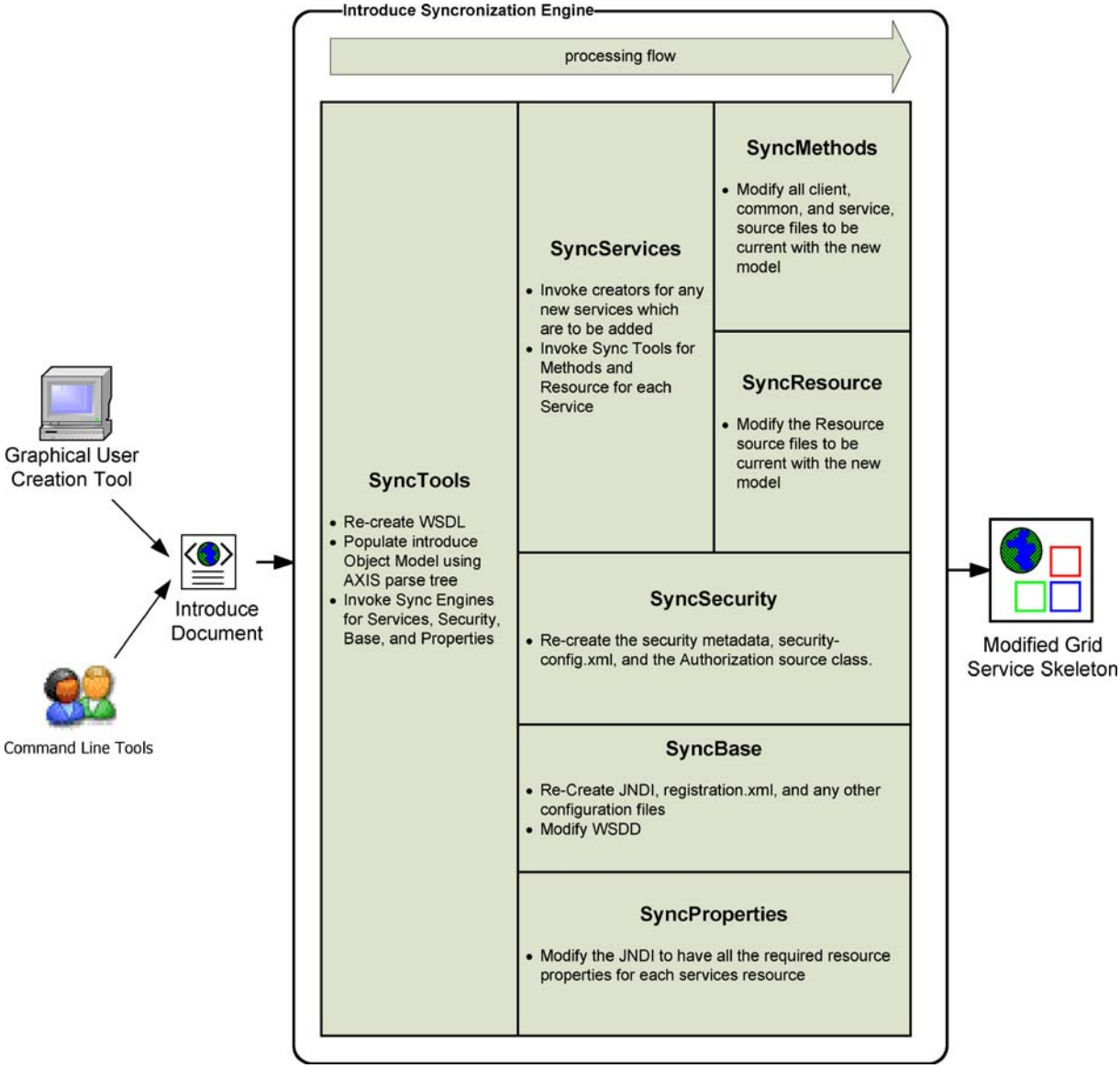

Fig. 4 Service resynchronization process

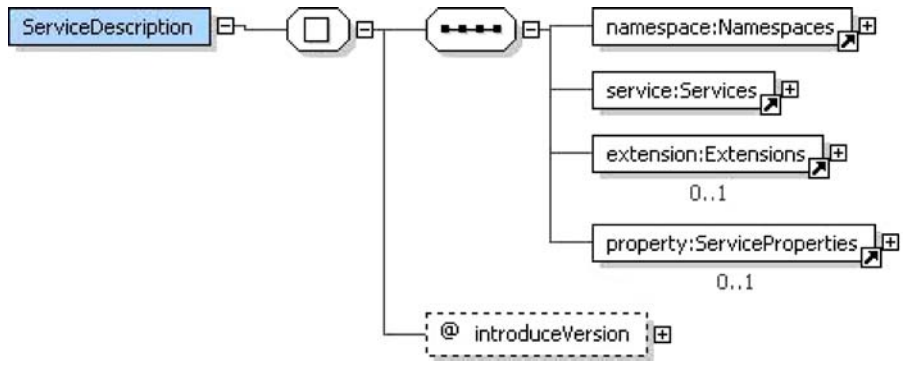

Fig. 5 Base Introduce service description schema

- Custom configuration files for IDE integration, e.g., Eclipse project files for editing the service using the Eclipse platform (http://www.eclipse.org),
- Standard interfaces 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, and
- 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 analyzes the service's current implementation with that of the desired service description. The overall high level process of service resynchronization is illustrated in Fig. 4. This process adds, removes, and modifies any service methods, resource properties, and service settings based on 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 (Figs. 5 and 6). The service description is the basis by which the code generation tools

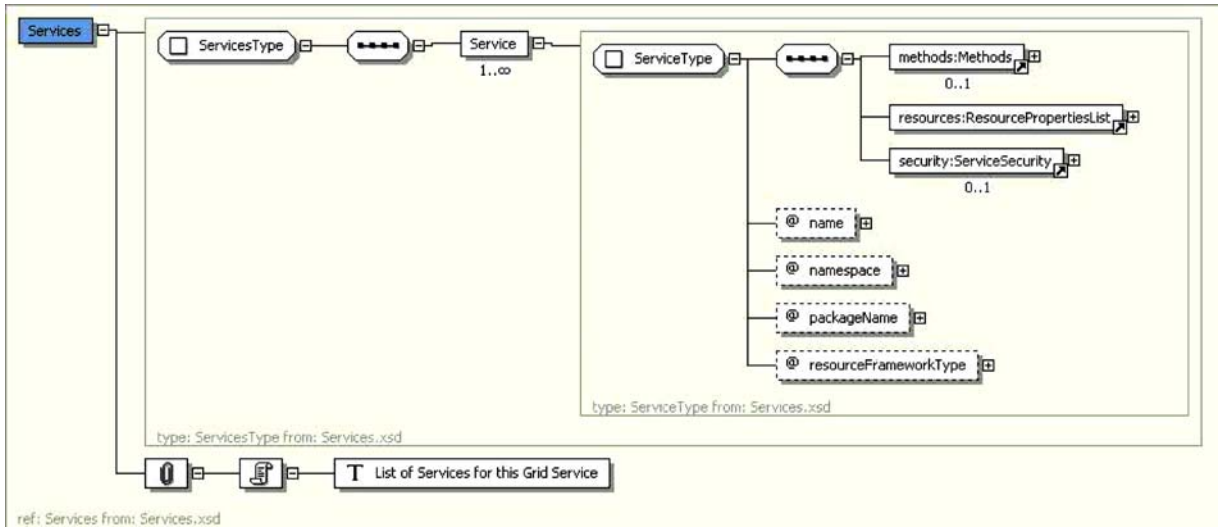

Fig. 6 Schema for Introduce services

add, remove, and modify operations and 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 uses the WSDL to generate the Grid stubs of the service.

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. 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 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 5.3). 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.

4.3.3 Service Deployer

The service deployment component currently supports deploying Grid services to a Globus or Tomcat container. The deployment framework, which is an ANT based deployer, can easily be extended to provide deployment capability to other Web Service containers as well as remote containers. It utilizes the layout of the Grid Archive (GAR) structure to organize and package a deployable service. The deployment process first populates template files (e.g., the WSDD and JNDI of the service) which specify 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 desired or 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 can be used to define configurable options for a particular deployment of the service. 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 or a code change prior to deployment.

5 Additional Features of Introduce Engine

In addition to basic service support, the Introduce engine has several additional components that implement features such as compositional inheritance, resource properties framework, multi-service/multi-resource contexts, security configuration, and auto-boxing/unboxing. In this section, we provide an overview of these components.

5.1 Auto-boxing/Unboxing of Service Operations

Code generation for Grid service interfaces has to associate the service interface as defined by the service developer to the actual port type generated by the GT via Axis. The overall process is complicated due to fact that the GT and Axis use document literal bindings to create the services portType and bindings to SOAP. For example, if the service developer describes a 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. 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 vice versa for the service (see Fig. 7). It is worth noting that Introduce created services can still be accessed via the standard document literal interfaces.

5.2 Resource Properties Framework

Introduce provides support for exposing service state and metadata provided by the WSRF-ResourceProperties (WSRF-RP) specification by abstracting away the details of common patterns of use. 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. For example, if the user wants to use a resource pattern which has the ability to control the lifetime of the resources which are created, the user will choose the resource lifetime style of resource pattern, and Introduce will generate a stubbed resource framework which supports the WS-ResourceLifetime specification. 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 the state or metadata of its resources by way of Resource Properties. Developers can either programmatically control the values of the property at runtime (as is common for exposed dynamic resource state), or supply the value from a file at service startup (as is common for exposed static metadata). Introduce also allows service developers to maintain soft-state registration of resource properties. The combination of these 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 instantiated by the container.

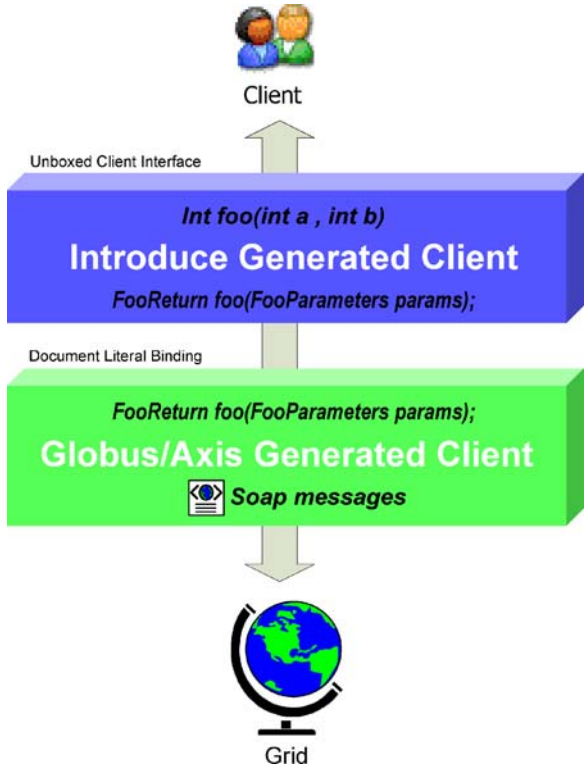

Fig. 7 Mapping from and to document literal port types and client/service level method syntax

5.3 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 as previously accomplished by the pre-WSDL based GT 3.2. 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 options 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 a stubbed method be generated 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 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 *operationProviders* variable of the WSDD configuration file of this service.

5.4 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 or several particular clients. These styles of Grid services use cases have driven the requirements for specifying a mechanism for stateful Grid services via the WSRF based specifications. Each WSRF service manages its state by creating and manipulating Resources. The main restriction is that a given service can only manage a single given resource type.

Consider a Grid data source, such as the one shown in Fig. 8, which provides access to a database. Assume this data source is designed to provide two capabilities: a "Query" capability, which allows a client to query into the backend database, and a "Results Delivery" capability, which provides the support to iteratively access query results (similar to a remote cursor). In a Grid service implementation of the data source, such capabilities are supported through the Service Context concept in Introduce. The "Query" capability is implemented as "Query Context" and the "Results Delivery" capability as "Results Delivery Context." 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

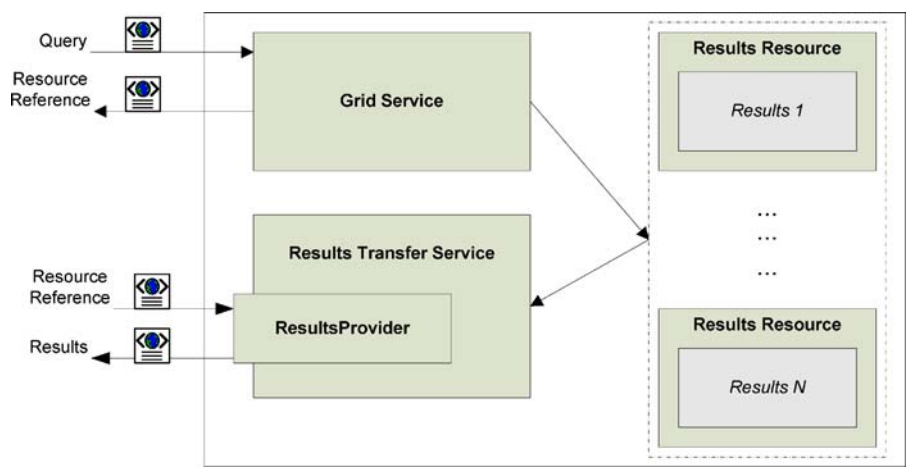

Fig. 8 Stateful Grid service pattern

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 Resource. The *query* operation then returns a pointer, or *EndPointReference* (EPR), based on the WS-Addressing specification, 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 Resource patterns for stateful Grid services.

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. Each created context has a corresponding source directory containing its own server, client, common, resource, etc. In this way, resource properties, operations, and security configurations can be added, removed, and modified for each additional context. The corresponding service and resource of each additional context are modified, compiled, and deployed with the main service.

5.5 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. Service developers can 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. In Introduce, 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 toolkit 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 policies 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 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.

6 Introduce Extension Framework

Introduce extension points enable custom plug-ins/extensions to be added to the Introduce engine to facilitate customization of services it can create. The Introduce Extension Framework currently consists of two styles of extensions; service and data type discovery. Extensions are added to the toolkit in the form of extension plug-ins, which the toolkit can then make it available to the service developer. 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.

6.1 Service Extensions

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: CreationUI, ModificationUI, CreationPostProcess, ModificationPreProcess, and ModificationPostProcess. Each of these extension components have a predefined class and interface that must be extended or implemented. Two of the service extension components, CreationUI and ModificationUI, are graphical components provided to the Introduce GDE, and the other three are Introduce engine plug-ins.

Each service extension component is invoked at a specific point in the creation or modification steps (Fig. 9). These different time points for each component execution are critical for making certain changes. For example, when the CreationUI 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 CreationUI component might prompt the user for particular information about the creation processes. This component would only be executed/displayed once for any given service and its non graphical component, the CreationPostProcess component, will also only be ran one time after the service has been created and before it will ever be modified.

The modification components are run 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 service description, 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.

6.2 Data Type Discovery Extensions

These extensions are Introduce GDE components that are accessed at the service modification step. They are

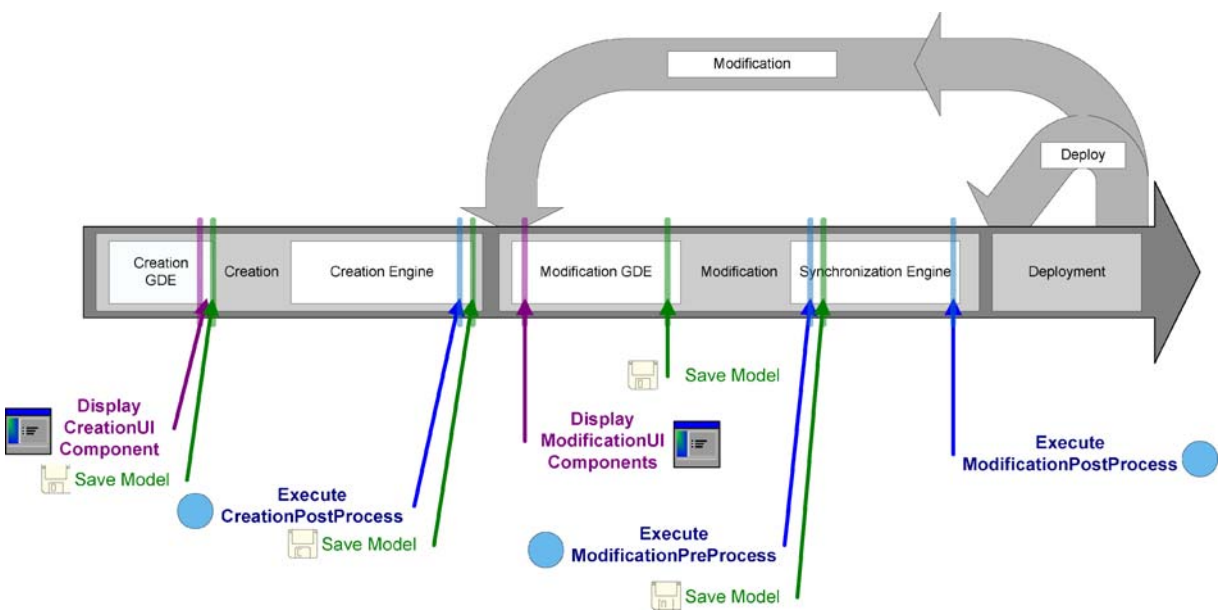

Fig. 9 Execution of Introduce extension components

intended to provide support for custom data type discovery for the service developer. They must allow the developer to browse types and chose to use those types in the developed service. This means that a 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.

7 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 heterogeneous 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. 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 and correct handling of data returned from services. (2) It allows for service developers to easily leverage advanced features of WSRF and the GT 4.0 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. Through use of plug-ins, this framework goes beyond simple

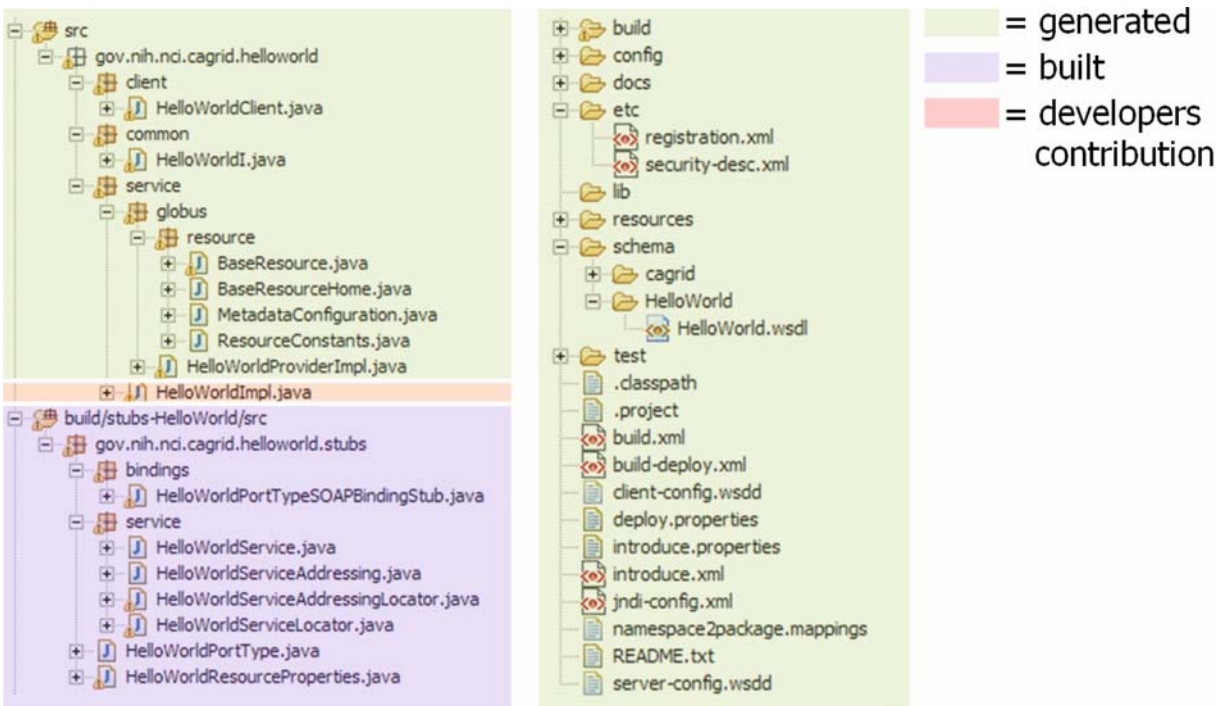

Fig. 10 Basic service layout of an example service created by Introduce

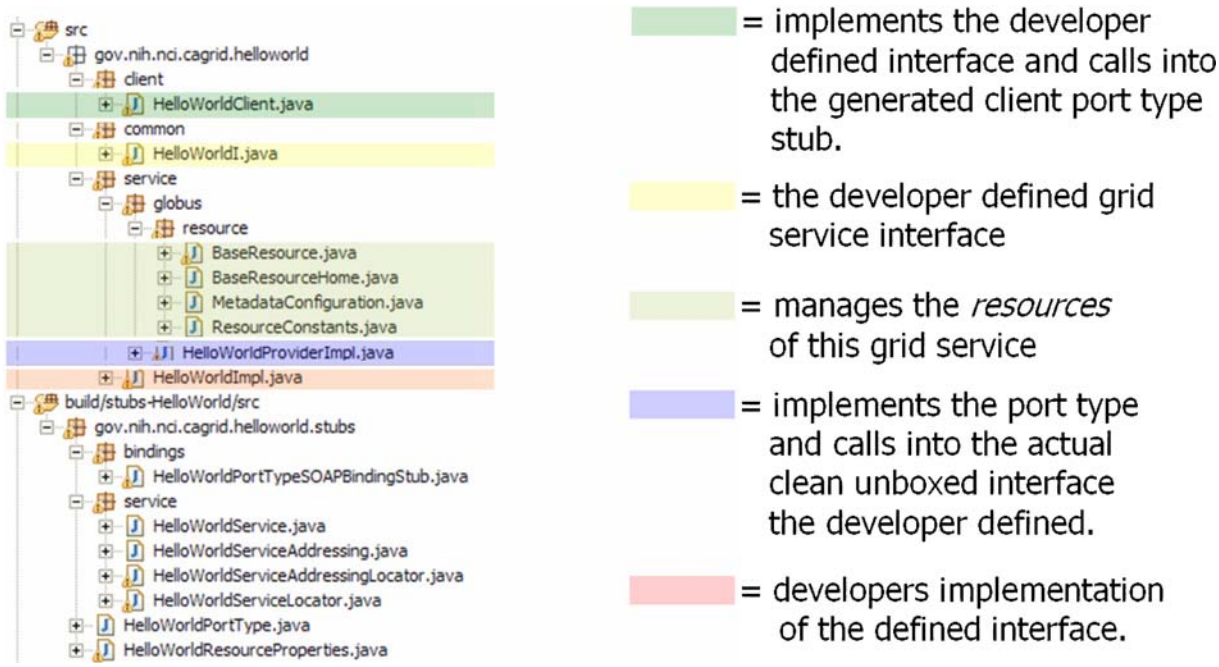

Fig. 11 Source files used in an example service created by Introduce

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

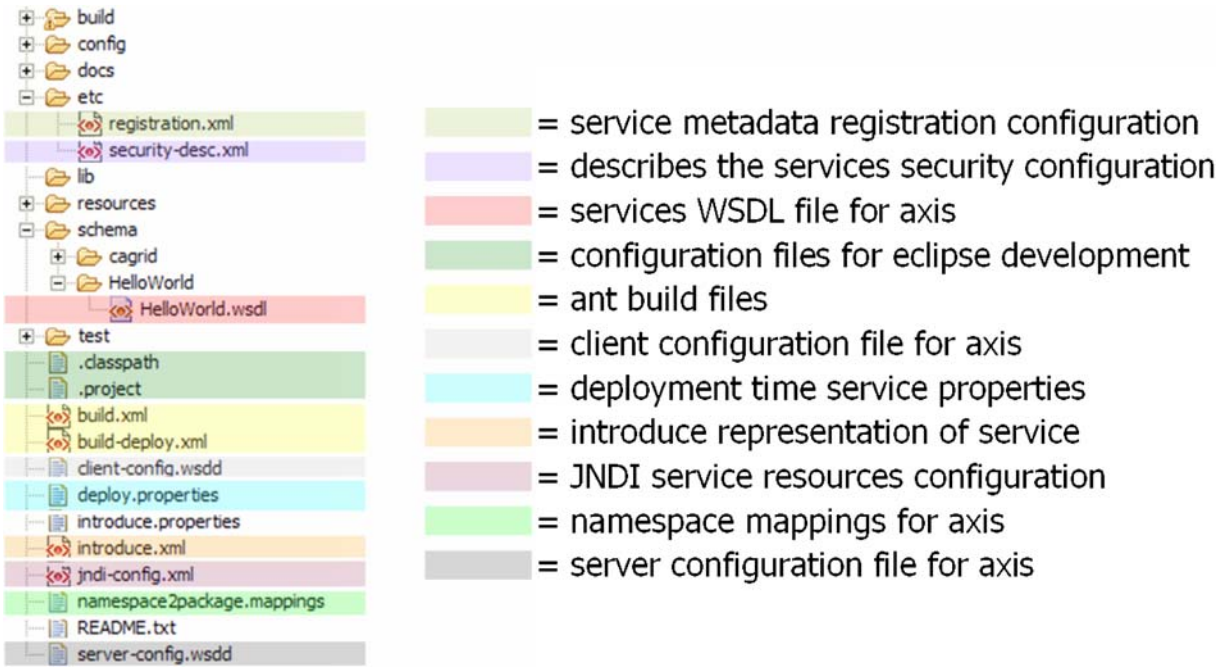

Fig. 12 Example common/configuration files on an example service created by Introduce

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 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.

Appendix

Examples of the Various Files and Directories Managed by Introduce for a Service

Figure 10 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 11 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 12 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 (http://www.eclipse.org).

References

1. Foster, I., Kesselman, C.: Globus: a metacomputing infrastructure toolkit. Int. J. High Perform. Comput. Appl. **11**, 115–128 (1997)
2. Foster, I., Kesselman, C., Tsudik, G., Tuecke, S.: A security architecture for computational Grids. In: Proceedings of the 5th ACM Conference on Computer and Communications Security Conference, pp. 83–92. ACM, New York (1998)
3. Welch, V., Siebenlist, F., Foster, I., Bresnahan, J., Czajkowski, K., Gawor, J., Kesselman, C., Meder, S., Pearlman, L., Tuecke, S.: Security for Grid services. In: 12th International Symposium on High Performance Distributed Computing (HPDC-12), 2003
4. Langella, S., Oster, S., Hastings, S., Siebenlist, F., Kurc, T., Saltz, J.: Dorian: Grid service infrastructure for identity management and federation. In: The 19th IEEE Symposium on Computer-Based Medical Systems, Special Track: Grids for Biomedical Informatics, Salt Lake City, Utah, 2006
5. Wolski, R., Spring, N., Hayes, J.: Network weather service: a distributed resource performance forecasting service for metacomputing. Future Gener. Comput. Syst. **15**, 757–768 (1999)
6. Czajkowski, K., Foster, I., Kesselman, C., Sander, V., Tuecke, S.: SNAP: a protocol for negotiating service level agreements and coordinating resource management in distributed systems. In: Job Scheduling Strategies for Parallel Processing, vol. 2537, pp. 153–183. Springer, Berlin Heidelberg New York (2002)
7. Frey, J., Tannenbaum, T., Livny, M., Foster, I., Tuecke, S.: Condor-G: a computational management agent for multi-institutional Grids. In: Proceedings of the Tenth Interna-

tional Symposium on High Performance Distributed Computing (HPDC-10). IEEE Press, Piscataway, NJ (2001)
8. Casanova, H., Graziano, O., Berman, F., Wolski, R.: The appLeS parameter sweep template: user-level middleware for the Grid. In: Proceedings of the ACM/IEEE Supercomputing Conference (SC2000). IEEE Computer Society Press, Los Alamitos, CA (2000)
9. Allcock, B., Bester, J., Bresnahan, J., Chervenak, A., Foster, I., Kesselman, C., Meder, S., Nefedova, V., Quesnal, D., Tuecke, S.: Data management and transfer in high performance computational Grid environments. Parallel Comput. **28**, 749–771 (2002)
10. Allcock, W.E., Foster, I., Madduri, R.: Reliable data transport: a critical service for the Grid. In: Proceedings of Building Service Based Grids Workshop, Global Grid Forum 11, Honolulu, HI, 2004
11. Chervenak, A., Deelman, E., Kesselman, C., Allcock, B., Foster, I., Nefedova, V., Lee, J., Sim, A., Shoshahi, A., Drach, B., Williams, D., Middleton, D.: High-performance remote access to climate simulation data: a challenge problem for data Grid technologies. Parallel Comput. **29**, 1335–1356 (2003)
12. Chervenak, A., Deelman, E., Foster, I., Guy, L., Hoschek, W., Iamnitchi, A., Kesselman, C., Kunst, P., Ripeanu, M., Schwartzkopf, B., Stockinger, H., Tierney, B.: Giggle: a framework for constructing scalable replica location services. In: Proceedings of the ACM/IEEE Supercomputing Conference (SC2002), pp. 1–17. IEEE Computer Society Press, Los Alamitos, CA (2002)
13. Ranganathan, K., Foster, I.: Identifying dynamic replication strategies for high performance data Grids. In: Proceedings of International Workshop on Grid Computing (GRID 2002) Denver, CO, 2002
14. Deelman, E., Singh, G., Atkinson, M.P., Chervenak, A., Chue Hong, N.P., Kesselman, C., Patil, S., Pearlman, L., Su, M.: Grid-based metadata services. In: Proceedings of the 16th International Conference on Scientific and Statistical Database Management (SSDBM '04), 2004
15. Singh, G., Bharathi, S., Chervenak, A., Deelman, E., Kesselman, C., Mahohar, M., Pail, S., Pearlman, L.: A metadata catalog service for data intensive applications. In: Proceedings of the ACM/IEEE Supercomputing Conference (SC2003), 2003
16. Hastings, S., Langella, S., Oster, S., Saltz, J.: Distributed data management and integration: the Mobius project. Proceedings of the Global Grid Forum 11 (GGF11) Semantic Grid Applications Workshop, Honolulu, HI, pp. 20–38, 2004
17. Allen, G., Dramlitsch, T., Foster, I., Goodale, T., Karonis, N., Ripeanu, M., Seidel, E., Toonen, B.: Cactus-G toolkit: supporting efficient execution in heterogeneous distributed computing environments. In: Proceedings of the 4th Globus Retreat Pittsburg, PA, 2000
18. Furmento, N., Lee, W., Mayer, A., Newhouse, S., Darlington, J.: ICENI: an open Grid service architecture implemented with JINI. In: Proceedings of the ACM/IEEE Supercomputing Conference (SC2002) Baltimore, MD. IEEE Computer Society Press, Los Alamitos, CA (2002)
19. Beynon, M., Kurc, T., Sussman, A., Saltz, J.: Design of a framework for data-intensive wide-area applications. In: Proceedings of the 9th Heterogeneous Computing Workshop (HCW2000), pp. 116–130. IEEE Computer Society Press, Los Alamitos, CA (2000)
20. Altintas, I., Bhagwanani, S., Buttler, D., Chandra, S., Cheng, Z., Coleman, M., Critchlow, T., Gupta, A., Han, W., Liu, L., Ludascher, B., Pu, C., Moore, R., Shoshani, A., Vouk, M.: A modeling and execution environment for distributed scientific workflows. In: Proceedings of the 15th International Conference on Scientific and Statistical Database Management (SSDBM '03) Boston, MA, 2003
21. Deelman, E., Blythe, J., Gil, Y., Kesselman, C., Mehta, G., Vahi, K., Blackburn, K., Lazzarini, A., Arbree, A., Cavanaugh, R., Koranda, S.: Mapping abstract complex workflows onto Grid environments. J. Grid Computing **1**, 25–39 (2003)
22. Foster, I., Voeckler, J., Wilde, M., Zhao, Y.: 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
23. Foster, I., Kesselman, C., Nick, J.M., Tuecke, S.: 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)
24. Foster, I., Kesselman, C., Tuecke, S.: The anatomy of the Grid: enabling scalable virtual organizations. Int. J. Supercomput. Appl. **15**, 200–222 (2001)
25. Cerami, E.: Web Services Essentials. O'Reilly (2002)
26. Graham, S., Simeonov, S., Boubez, T., Davis, D., Daniels, G., Nakamura, Y., Neyama, R.: Building Web Services with Java: Making Sense of XML, SOAP, WSDL, and UDDI. SAMS Publishing (2002)
27. Czajkowski, K., Ferguson, D.F., Foster, I., Frey, J., Graham, S., Sedukhin, I., Snelling, D., Tuecke, S., Vambenepe, W.: The WS-Resource Framework Version 1.0. http://www.globus.org/wsrf/specs/ws-wsrf.pdf (2004)
28. Hastings, S., Langella, S., Oster, S., Kurc, T., Pan, T., Catalyurek, U., Janies, D., Saltz, J.: Grid-based management of biomedical data using an xML-based distributed data management system. In: Proceedings of the 20th ACM Symposium on Applied Computing (SAC 2005), Bioinformatics Track, Santa Fe, New Mexico. ACM, New York (2005)
29. Langella, S., Hastings, S., Oster, S., Kurc, T., Catalyurek, U., Saltz, J.: A distributed data management middleware for data-driven application systems. In: Proceedings of the 2004 IEEE International Conference on Cluster Computing (Cluster 2004), 2004
30. Saltz, J., Oster, S., Hastings, S., Kurc, T., Sanchez, W., Kher, M., Manisundaram, A., Shanbhag, K., Covitz, P.: caGrid: design and implementation of the core architecture of the cancer biomedical informatics Grid. Bioinformatics **22**(15), 1910–1916 (2006)
31. Smith, M., Friese, T., Freisleben, B.: Model driven development of service oriented Grid applications. In: Advanced International Conference on Telecommunications and International Conference on Internet and Web Applications and Services (AICT-ICIW '06), 2006

32. Friese, T., Smith, M., Freisleben, B.: Grid development tools for eclipse. In: Eclipse Technology eXchange workshop (eTX) at ECOOP 2006, Nantes, France, 2006
33. Morohoshi, H., Huang, R.: A user-friendly platform for developing grid services over globus toolkit 3. In: The 2005 11th International Conference on Parallel and Distributed Systems (ICPADS'05), 2005
34. Mizuta, S., Huang, R.: Automation of Grid service code generation with AndroMDA for GT3. In: The 19th International Conference on Advanced Information Networking and Applications (AINA'05), 2005
35. von Laszewski, G., Foster, I., Gawor, J., Lane, P.: A Java commodity grid kit. Concurr. Comp.-Pract. E. **13**, 643–662 (2001)
36. von Laszewski, G., Foster, I., Gawor, J., Smith, W., Tuecke, S.: CoG kits: a bridge between commodity distributed computing and high performance Grids. In: ACM Java Grande 2000 Conference, 2000
37. Humphrey, M., Wasson, G.: Architectural foundations of WSRF .NET. JWSR **2**, 83–97 (2005)
38. Kacsuk, P., Doza, G., Kovacs, J., Lovas, R., Podhorszki, N., Balaton, Z., Gombas, G.: P-GRADE: a Grid programming environment. J. Grid Computing **1**, 171–197 (2003)
39. Fahringer, T., Jugravu, A., Pllana, S., Prodan, R., Seragiotto, C. Jr., Truong, H.-L.: ASKALON: a tool set for cluster and Grid computing. Concurr. Comp.-Pract. E. **17**, 143–169 (2005)
40. Delaitre, T., Kiss, T., Goyeneche, A., Terstyanszky, G., Winter, S., Kacsuk, P.: GEMLCA: running legacy code applications as Grid services. J. Grid Computing **3**, 75–90 (2005)