Service-Oriented Architecture and Technologies for Automating Integration of Manufacturing Systems and Services

Yi Zhi Zhao, Jing Bing Zhang, Liqun Zhuang, Dan Hong Zhang
Singapore Institute of Manufacturing Technology
71 Nanyang Drive, Singapore 638075
yzzhao@simtech.a-star.edu.sg

Abstract

Enterprise application integration (EAI) has been pursued strenuously by researchers and industrial practitioners for decades. With the advent of Web Services and Service-Oriented Architecture (SOA), realizing enterprise integration, even automating inter-enterprise interactions become feasible. Virtualization is the fundamental for implementing an SOA based on Web Service technologies. In this paper, we first review the Web Service-Resource Framework (WS-RF) with emphasis on the concepts of Web Services and stateful resource modeling. Then we outline a Service-Oriented architecture that shows how various types of manufacturing resources are virtualized as Web Services; what technologies are used in Web Services register, discovery and transport, and in business process automation. We adopt the WS-Resource approach, for illustration purpose, to virtualize a category of resource-Automatic Guided Vehicle (AGV). Different projections are defined based on WS-Resource properties upon virtualization so as to offer different perspectives to the same resources to meet a variety of business requirements.

1. Introduction

Enterprise application integration (EAI) has been pursued strenuously by researchers and industrial practitioners for decades. The most renowned network computing, middleware and relevant technologies used for EAI could not provide a complete solution to holistic enterprise integration, but have contributed more or less to the development and emergence of more promising technologies. They include DCE (Distributed Computing Environment) [1, 2], CORBA (Common Object Request Broker Architecture) [3], IIOP (Internet Inter-ORB Protocol – communication protocol within CORBA), DCOM (Microsoft's Distributed Component Object Model), Java RMI (Java Remote Method Invocation) [4], UML (Unified Modeling Language), as well as Java 2 Platform, Enterprise Edition (J2EE) [5, 6]. Table 1 shows the different data format, interface language and transfer protocol that are used by these middleware technologies.

Each of these network computing and middleware technologies has defined its own vertical set of formats and protocols in order to implement the core functions required. This approach ensures consistency among applications that share the same middleware, but prevents interoperability with applications that use different middleware. It also requires service producers and service consumers that wish to engage in a conversation to have the appropriate middleware loaded on their machines. For example, if you are using CORBA, infrastructure such as an Object Request Broker (ORB) must be installed on the host computer for a system to resolve the CORBA IDL types. Two CORBA-based systems that are installed on different hardware platforms and operating systems can interoperate, but they both remain coupled to the CORBA type specification and binary mappings. Such homogeneous applications create silos, which are themselves maybe of strong power and smooth in communication, but cannot understand one another. Therefore; holistic enterprise integration has yet to become reality mainly because of the constraints that are posed by the preceding technologies.

Table 1: A comparison of format, interface and protocol of middleware technologies

	DCE	**CORBA**	**Java RMI**	*Web Services*
Data Format	NDR (Network Data Representation)	CDR (Common Data Representation)	Serialized Java	*XML*
Wire Format	PDU (Protocol Data Units)	GIOP (General Inter-ORB Protocol)	Stream	*SOAP*
Interface Description	DCE IDL (Interface Definition Language)	CORBA IDL (Interface Definition Language)	Java interface	*WSDL*
Transfer Protocol	RPC CO (Connect-Oriented protocol)	IIOP (Internet Inter-ORB Protocol)	JRMP (Java Remote Method Protocol)	*HTTP, SMTP*
Discovery Mechanism	CDS (Cell Directory Service)	COS naming (CORBA Object Service)	Java Registry	*UDDI*
Invocation Mechanism	RPC (Remote Procedure Call)	CORBA RMI (remote Method Invocation)	Java RMI (Remote Method Invocation)	*JAX-RPC, .NET*

Unlike its predecessors, Web Services does not require an entirely new set of protocols. The most basic

0-7803-9402-X/05/$20.00 © 2005 IEEE

Web Services protocol is the industry standard eXtensible Markup Language (XML), which is used not only as the message data format, but also as the foundation for all other Web Services protocols. Web Services system implementation is based on a core set of technologies that include SOAP (Simple Object Access Protocol), WSDL (Web Services Description Language) and UDDI (Universal Description, Discovery and Integration) which define the transport, interface description and discovery mechanism respectively. Each of these technologies is defined and implemented in XML. The crucial point here is that any application, written in any language, running on any platform, can interpret Web Services messages, interface description and discovery mechanisms that are implemented in XML using standard XML processing tools without the need for any specific middleware to be used.

With the advent of Web Services and Service-Oriented Architecture (SOA), realizing enterprise integration, accelerating enterprise responsiveness to customers, automating inter-enterprise interactions and optimizing the business processes of the whole supply chain become feasible. The service-oriented paradigm and Web Services technologies are rapidly emerging as the most practical approaches for integrating a wide array of manufacturing resources in the manufacturing grid environment, and efforts in the Semantic Web standards and technologies present an opportunity for automating the integration process. Web Services are the most likely to be adopted as the de facto standard to deliver effective, reliable, scalable and extensible machine-to-machine interaction than any of its predecessors.

Virtualization is the fundamental for implementing an SOA based on Web Service technologies. In the SOA approach, all resources, including various data sources, manufacturing devices/equipment/subsystems, processes/applications, manpower, etc, are required to be virtualized and exposed as Web Services [15]. Upon being virtualized and exposed as Web Services, these components and resources can present their functionalities and transform manufacturing resources into service capabilities in a standard Web Services environment. With service brokering, orchestration and choreography under the WS-RF, automating the integration of manufacturing execution and services can be achieved.

2. Web Services-Resource Framework (WS-RF) Review

To understand how WS-RF can provide stateful services, we must first understand what Web Services are and how a state is defined in a standard manner.

2.1. Web Services and Stateful Resource

According to the World Wide Web Consortium (W3C) Web Services architecture working group, a Web Service is defined as "*a software system designed to support interoperable **machine-to-machine interaction** over a network, and it has an **interface described in a machine-processable format** (specially WSDL), and other systems **interact with the web service** in a manner prescribed by its description **using SOAP-messages**, typically conveyed using HTTP with an **XML serialization** in conjunction with other web-related standards*" [7].

It can be seen that Web Services is a collection of Web technologies, including

XML: a platform-neutral mechanism to represent data.

SOAP: the data communication protocol for Web Services.

WSDL: a definition language for describing a Web Service and its specific capabilities.

UDDI: a means to advertise and discover Web Services.

While Web Services have been successfully used to develop applications for human-to-human communication and human-to-machine communication since 2000, they mainly serve as "*stateless*" and "*non-transient*" services. "*Stateless*" means that they do not retain the results that are returned after being invoked. "*Non-transient*" means that they outlive all their clients. As "state" appears in almost all applications, such as data in a purchase order (shopping cart being one of the best-known examples of stateful applications on the Web), current state of resources and current usage agreement for resources, etc., it is a necessity to handle "state" in a native and efficient way. In Web Services, the use of tools such as databases, XML files, cookies, and session handling must be combined to add "statefulness" to the applications - outside the HTTP protocol to reach this goal. In the shopping cart case, they keep track of what you add or remove from the shopping cart, either with cookies or server-side session software. The data values are stored from page view to page view and even evolve as the interaction increases (for example, the total price goes up as you add more things to the cart). Undoubtedly this is not a native way to handle state, and will encumber application development with additional functionalities.

GGF's OGSI (Global Grid Forum's Open Grid Services Infrastructure) V1.0 specification defined a set of approaches for modeling, accessing and managing state. The approaches are used for creating, naming, and managing the lifetime of instances of services; for declaring and inspecting service state data; for asynchronous notification of service state change; for representing and managing collections of service instances; and for common handling of service invocation faults [8]. Although the Grid Service

overcomes Web Service's *stateless* and *non-transient* limitations by introducing a service factory that allow you to create service instances so that state can be encapsulated inside the service, this approach has not been widely accepted by the industry yet. The main problem brought about by this mechanism is that deploying a new Grid service to the hosting environment requires a restart of the service container, which in turn compels all the ongoing services and persistent services to stop (e.g. Globus Toolkit3). Some other criticisms of OGSI from Web Services community include: 1) "[It contains] *too much stuff in one specification"*; 2) "[It] *does not work well with existing Web Services and XML tooling"*; 3) "[It is] *too object oriented"*; etc. [8].

2.2. Modeling Stateful Resources with Web Service

The WS-Resource Framework (WS-RF) was proposed as a refactoring and evolution of OGSI in a manner that exploits recent developments in Web Services architecture (particularly WS-Addressing). The framework provides the means to express state as *stateful resources* and codifies the relationship between Web Services and stateful resources in terms of the *implied resource pattern* [9]. A *stateful resource* can be defined as a component that has three characteristics [9]: (1) [It is] composed by a specific set of state data expressible as an XML document; (2) [It] has a well-defined lifecycle; and (3) [It] can be manipulated by one or more Web Services. *Implied resource pattern* refers to the mechanisms used to associate a stateful resource with the execution of message exchange implemented by a Web Service [9].

WS-Addressing provides and standardizes the endpoint reference construct used to represent the address of a Web Service deployed at a given network endpoint. An *endpoint reference* is represented as an XML serialization, usually returned as a result of a Web Service request to create a new resource, or as a result of some application-specific Web Service request. An endpoint reference might contain, besides the Web Service address, metadata such as service description and reference properties. Figure 1 shows how a Web Service is associated with a stateful resource in a WS-Addressing endpoint reference and how it is used in message exchange.

The key steps involved in the creation of an endpoint reference includes: (1) the Service Requestor sends a request to the Message Processing Facility (MPF, which may support one or more network transport protocols, such as HTTP, SMTP or IIOP) in runtime environment E; (2) MPF identifies and pass the message to the wanted service *WebServiceX*; (3) *WebServiceX* creates a new stateful resource, and assigns the resource a unique identifier R1 (resource ID) within the service context; (4) *WebServiceX* creates the association between *WebServiceX* and the stateful resource *R1* and pass to MPF; and (5) MPF returns an endpoint reference EPR that contains the association to Service Requestor. The endpoint reference contains two important components:

- The `<wsa:Address>` component refers to the network transport-specific address of the Web Service (often a URL in the case of HTTP-based transports).

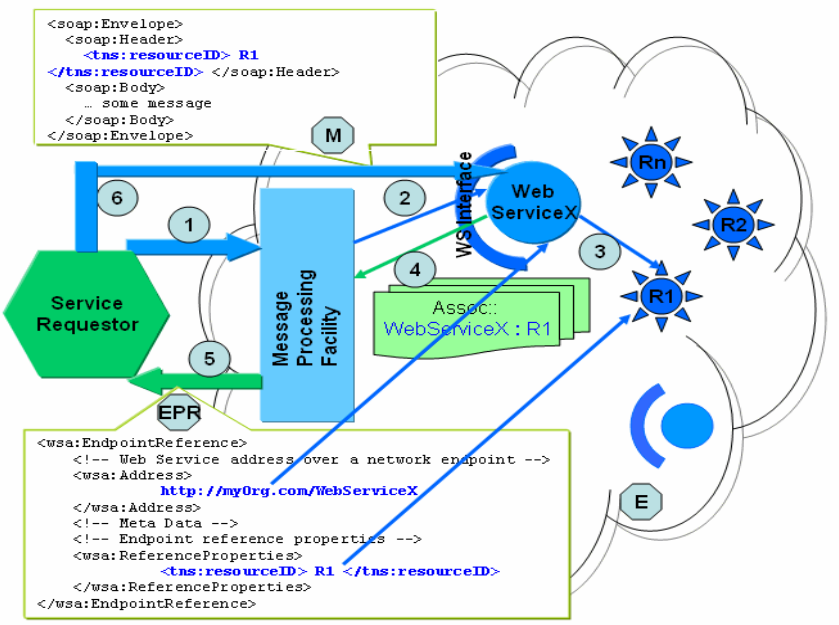


Figure 1: An Endpoint Reference containing a stateful resource identifier and used in message change

- The `<wsa:ReferenceProperties>` component may contain an XML serialization of a stateful resource identifier, which will be used in the execution of the request message.

(6) Upon receiving the endpoint reference EPR, Service Requestor's application would use the EPR to send message M to the identified *WebServiceX* through its interface. Message M shows how to use the SOAP header to propagate the stateful resource identifier R1. The *WebServiceX* then extracts the R1 from the SOAP message and uses it to locate the stateful resource needed for the execution of the request message.

WS-Addressing provides a mechanism to separate state management of a stateful resource from the Web Services associated with it without compromising the ability to implement the Web Services as stateless message processor. In consequence, the joining and leaving of a service does not require a restart of the service container. Therefore; it has no impact on the ongoing services and persistent services.

The composition of a stateful resource and a Web Service that participates in the implied resource pattern is termed a *WS-Resource* [8]. A WS-Resource can be created and destroyed, and their state can be queried and modified via Web Service message exchanges. WS-Resource has four characteristics, known as ACID properties that are explained briefly below [14].

- Atomicity**:** Stateful resource updates within a transactional unit are made in an all-or-nothing fashion.
- Consistency**:** Stateful resources should always be in a consistent state even after failures.
- Isolation**:** Updates to stateful resources should be isolated within a given transactional work unit.
- Durability**:** Provides for the permanence of stateful resource updates made under the transactional unit of work.

3. Architecture for Automating Integration of Manufacturing Systems and Services

The above WS-Resource-based approach provides an enabling technology for modeling the resources of manufacturing systems in a service-oriented paradigm. In a service-oriented paradigm, all resources, including various data sources, manufacturing devices/ equipment/ subsystems, processes/ applications, manpower, etc., are required to be virtualized and exposed as Web Services. Upon being virtualized and exposed as Web Services, these components and resources can present their functionalities and transform manufacturing resources into service capabilities in a standard Web Services environment. With service brokering, orchestration and choreography under WS-Resource Framework (WS-RF), automating the integration of manufacturing execution and services can be achieved. Figure 2 shows a service-oriented architecture for automating the integration of manufacturing systems and services.

3.1. Manufacturing Systems and Resources

The components in this layer represent the existing resources and systems in a manufacturing environment. They might be database systems, file systems, mainframe servers, workflows, processes, applications, manufacturing conveyer systems like AGV, storage systems like Automated Storage/Retrieval System (AS/RS), and manpower, etc.

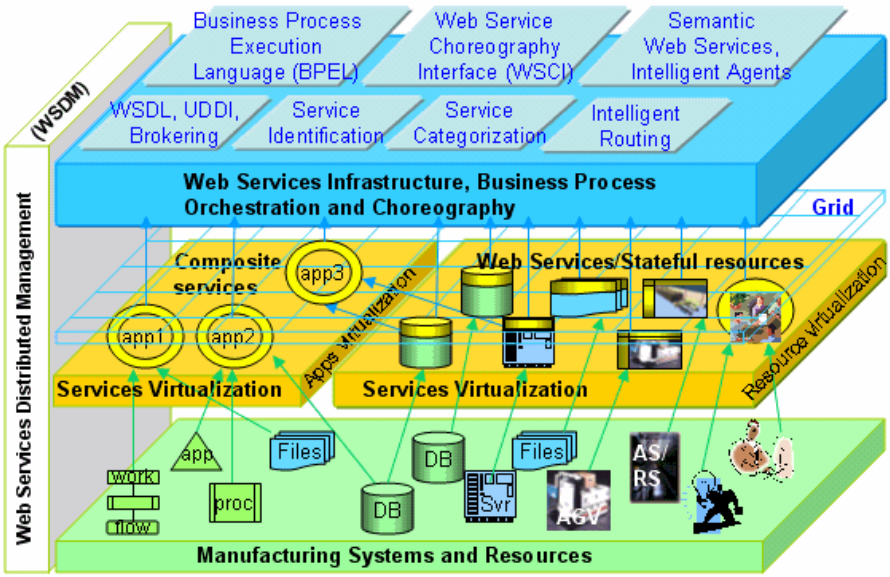


Figure 2: Service-oriented architecture for automating the integration of manufacturing systems and services

3.2. Services Virtualization

All the components in manufacturing systems and resources layer are virtualized and exposed as Web Services based on the WS-Resource approach. *Resource virtualization* provides an abstraction that allows applications to access physical resources indirectly, without being tied directly to specific physical resources (such as servers or database). *Application virtualization* allows business functionalities to be hosted as application services that could be accessed whenever they are required without requiring a physical system endpoint to be known or even needed to be executed.

A virtualized application is capable of not only remotely invoking application requests and returning results, but also ensuring that the application state and other data are available and consistent on all resource nodes that are executing the application across an enterprise. From the viewpoint of WSDM and the top layer, all virtualized resources and applications will construct a *manufacturing grid* where they can be invoked through open standard Web service message exchange in WS-RF.

3.3. Web Service Infrastructure and Business Process Orchestration and Choreography:

The three core functional components (transport, description, and discovery) are fundamentals of the Web Services infrastructure, and they are implemented using SOAP, WSDL and UDDI, respectively. A UDDI registry plays the role of service broker. The *register* and *find* operations are implemented using the UDDI Inquiry and UDDI Publish APIs. A WSDL document describes the service contract, and is used to bind the client to the service. All transport functions are performed using SOAP.

The *Service Identification* process usually consists of a combination of top-down and bottom-up techniques of domain decomposition, existing asset analysis and goal-service modeling. In the top-down view, business use cases provide the specification for business services. This top-down process is often referred to as domain decomposition, which consists of the decomposition of the business domain into its functional areas and subsystems, as well as high-level business use cases. These use cases often are not only very good candidates for business services exposed at the edge of the enterprise, but also used within the enterprise boundaries across lines of business. In the bottom-up approach, existing systems are analyzed and selected as viable candidates for providing lower cost solutions to the implementation of underlying service functionality that supports the business process.

The *Service Categorization* allows the service provider to categorize the entity (business or service) using a variety of taxonomies. The UDDI specification defines a core set of taxonomies, such as geographic location, product codes and industry codes. Additional taxonomies can be added to the registry to support more focused or customized categorization and search. Through *service classification*, Web Services are classified into a service hierarchy according to their granularity in order to facilitate service composition and efficient routing.

Compositions and Choreographies of Services exposed in the service virtualization layer are defined using Business Process Execution Language (BPEL) and Web Service Choreography Interface (WSCI) specification in this layer. Services are bundled into a flow through orchestration or choreography, and thus act together as a single application. These applications support specific use cases and business processes.

The Web Services in this SOA model will be managed according to the emerging industry standard Web Services Distributed Management (WSDM from OASIS). There are two aspects of management involved in WSDM: Management Using Web Services (MUMS) and Management Of Web Services (MOMS). MUMS defines how a resource in a SOA-based manufacturing environment provides the manageability interfaces such that it can be remotely managed using Web Services technologies. MOMS defines management of any IT resources via Web Services protocols and management of the Web Services resources.

4. Service Virtualization based on WS-Resource Approach

Service Virtualization based on the WS-Resource approach is a set of transformation processes by which associations between the virtualized entities and underlying components are established using WS-Resource Properties according to business requirements. The underlying components can be any resource within an enterprise. For illustration purpose, a typical type of material handling system AGV in a manufacturing system has been virtualized as Web service.

An AGV is a class of material handling equipment with the following characteristics [10]:

- Each vehicle is self-propelled. On-board batteries supply power to electric motors that drive and steer the vehicle.
- Pathways are to be defined in the factory or warehouse. Navigation technologies include fixed track, electromagnetic field, and sensor guide movement.
- Types of AGV include: towing vehicle, fork lifts, pallet trucks, and unit load carriers. They might be supplied from different vendors and with different performance and features.

Holistic integration of manufacturing systems and services for different business purposes needs multiple views on, or projections of, the actual components. For an AGV, its multiple views can be identified and described as follows:

- *Physical View:* defining the AGV's type, guidance type, load type, load transfer type, battery type, weight, curve radius, max operation speed, etc.
- *Operational View:* reflecting the AGV's snapshot in the actual operation time by capturing its original/ destination station, current zone/position, next zone, current load/operation mode/speed/shift number, distance travelled/running time in current shift, etc.
- *Performance View:* depicting the AGV's running time percentage, utilization percentage, ideal time, down time, congestion time, availability, total weight carried, number of job performed, etc.
- *Finance View:* accounting for the AGV's direct investment cost, installation/transportation/ maintenance cost, charge/depreciation rates, payback time, etc.

Services virtualization based on the WS-Resource approach provides multiple views by using WS-Resource properties document (WS-ResourceProperties). The resource properties document type associated with a Web Service's WSDL 1.1 portType definition provides the declaration of the exposed resource properties of the WS-Resource [11]. An Operational View was taken as an illustrative example for virtualisation as shown in Figure 3. The View captured from the Web browser is shown in Figure 4. Service virtualization for the other views can be done similarly.

```
<wsdl:definitions …
    xmlns:tns=http://localhost/agvviews …>
    …
    <wsdl:types>
        <xsd:schema targetNamespace=" http://localhost/agvviews">
            <!-- Resource property element declarations -->
            <xsd:element name="AGVOriginalStation" type="xsd:string"/>
            <xsd:element name="AGVDestinationStation" type="xsd:integer"/>
            <xsd:element name="AGVCurrentPosition" type="xsd:string"/>
            <xsd:element name="AGVCurrentzone" type="xsd:string"/>
              … …
            <xsd:element name="AGVModeOfOperation" type="xsd:string"/>
            <xsd:element name="AGVJobPriority" type="xsd:integer"/>
              … …
            <!-- Resource properties document declaration -->
            <xsd:element name="AGVOperationalViewProperties">
              <xsd:complexType>
                  <xsd:sequence>
                      <xsd:element ref="tns:AGVOriginalStation"/>
                      <xsd:element ref="tns:AGVDestinationStation" />
                      <xsd:element ref="tns:AGVCurrentPosition" />
                      <xsd:element ref="tns:AGVCurrentZone" />
                          … …
                      <xsd:element ref="tns:AGVModeOfOperation" />
                      <xsd:element ref="tns:AGVJobPriority" />
                          … …
                  </xsd:sequence>
              </xsd:complexType>
            </xsd:element>
              … …
        </xsd:schema>
    </wsdl:types>
    … …
    <!-- Association of resource properties document to a portType -->
    <wsdl:portType name="AGVOperationalView"
        wsrp:ResourceProperties="tns:AGVOperationalViewProperties" >
    … …
    </wsdl:portType>
    … …
</wsdl:definitions>
```

Figure 3: Operational view of AGV virtualized as Web Service

Operational details of the AGV 1111

AGV ID	1111
Originating Station	Station03
Destination Station	Station02
Current Position	Station04
Current Zone	Zone05
Previous Zone	Zone07
Next Zone	Zone08
Waiting Time At Current Zone	00:07:00 hrs
Waiting Time At Previous Zone	00:02:00 hrs
Lane Id	Lane05
Current Load	40.00 kgs
Current Mode Of Operation	Automatic
Current Speed	30.00 m/s
Current Shift Number	4
Job Priority	3
Vehicle Priority	2
Type Of Operation Path	FreePath
Distance Travelled In Current Shift	55.00 m
Running Time In Current Shift	01:20:00 hrs

Figure 4: Operational view of AGV in Web browser

5. Conclusion and Future Work

In this paper, we have presented the WS-Resource Framework with focus on the concepts of Web Services and stateful resource modeling with Web Services. Then we outlined a Service-Oriented Architecture that shows how various types of manufacturing resources are virtualized as Web Services; the technologies that are used in the Web Services register, discovery and transport, and in business process automation. We adopted the WS-Resource approach to virtualize a typical material handling equipment resource - an AGV as a Web Service. Different projections are defined based on WS-Resource properties upon virtualization so as to offer different perspectives to the same resources to meet a variety of business requirements.

Service-Oriented Architecture, WS-RF, and their underlying standards, specifications are still evolving. This provides opportunities for pioneering researchers and practitioners to explore the new technologies and mechanisms to address the yet–to-be-resolved challenges facing the enterprise application integrator as highlighted below:

1) For service virtualization, what are the mechanisms for automatic generation of WSDL for the new virtual services based on existing services? In other words, how to realize automatic service scaled up?

2) For service brokering, how to support sophisticated load-balancing and failover so as to ensure quality of services (QoS) in the most satisfactory degree of service level agreement (SLA)?

The needs to coordinate geographically or logically distributed manufacturing resources based on open standard protocol to deliver agreed quality of service are increasing dramatically. We believe that they must inspire international communities and researchers to transform the traditional manufacturing network architecture into a novel manufacturing grid architecture that dynamically harnesses these significant, heterogeneous manufacturing capabilities and "*autonomous*" resources in order to satisfy dynamic business requirements.

6. Acknowledgement

This work (project code U04-E-296A) was supported in part by the Agency for Science, Technology and Research (A*STAR) under The Thematic Strategic Research Program "Integrated Manufacturing and Service Systems (IMSS)".

References

[1] Ward Rosenberry, David Kenney, & Gerry Fisher, *Understanding DCE*, O'Reilly & Associates, Inc., 1992, ISBN 1-56592-005-8.

[2] John Shirley, *Guide to Writing DCE Applications*, O'Reilly & Associates Inc., 1992, ISBN 1-56592-004-X.

[3] Object Management Group, *The Common Object Request Broker: Architecture and Specification*, *revision 2.0, updated July 1996.*

[4] William Grosso, *Java RMI*, O'Reilly, October 2001, ISBN: 1-565-92452-5.

[5] Bill Shannon, Mark Hapner, Vlada Matena, James Davidson, Larry Cable, *Java 2 Platform, Enterprise Edition: Platform and Component,* Addison-Wesley, May 2000, ISBN: 0-201-70456-0.

[6] Nicholas Kassem, *Designing Enterprise Applications with the Java 2 Platform, Enterprise Edition*, Addison-Wesley, October 2000, ISBN: 0-201-70277-0.

[7] "Web Services Architecture W3C Working Group Note 11 February 2004." Available: *http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/*

[8] K. Czajkowski, D. Ferguson, I. Foster, J. Frey, S. Graham, D. Snelling, and S. Tuecke. "From Open Grid Services Infrastructure to Web Services Resource Framework: Refactoring and Evolution". Global Grid Forum. *http://www-106.ibm.com/developerworks/library/ws-resource/ogsi_to_wsrf_1.0.pdf*.

[9] Ian Foster, Jeffrey Frey, Steve Graham, Steve Tuecke, Karl Czajkowski, Don Ferguson, Frank Leymann, Martin Nally, Igor Sedukhin, David Snelling, Tony Storey, William Vambenepe, and Sanjiva Weerawarana. "Modeling Stateful Resource with Web Services", v1.1, 3 May 2004. *http://www-106.ibm.com/developerworks/library/ws-resource/ws-modelingresources.html*

[10] Material Handling Industry's online elessons: *http://www.mhia.org/et/et_mhi_elessons_home.cfm*

[11] Steve Graham, Jem Treadwell (eds.), "Web Services Resource Properties v1.2 (WS-ResourceProperties)". OASIS WSRF TC, Working Draft 06, 5 April 2005. *http://docs.oasis-open.org/wsrf/2005/03/wsrf-WS-ResourceProperties-1.2-draft-06.pdf*

[12] Karl Czajkowski, Donald F Ferguson, Ian Foster, Jeffrey Frey, Steve Graham, Igor Sedukhin, David Snelling, Steve Tuecke, and William Vambenepe. "The WS-Resource Framework v1.0", 3 May 2004. *http://www-106.ibm.com/developerworks/library/ws-resource/ws-wsrf.pdf*.

[13] W3C. "Web Services Addressing (WS-Addressing)". W3C Member Submission 10 August 2004: *http://www.w3.org/Submission/2004/SUBM-ws-addressing-20040810/*.

[14] Luis Felipe Cabrera, George Copeland, Max Feingold, Tom Freund, et al., "Web Services Atomic Transaction (WS-AtomicTransaction)", November 2004. http://www-128.ibm.com/developerworks/library/specification/ws-tx/.

[15] Rajaram Shivram Kumar, Zhonghua Yang, Jing Bing Zhang, and Liqun Zhuang, "Virtualization for Manufacturing Web Services: WS-RF Approach", The 2nd Workshop on Grid Computing & Applications, Grid Asia 2005, pp. 38-47, Biopolis, Singapore May 2-6, 2005.

[16] S. Tuecke, K. Czajkowski, I. Foster, J. Frey, S. Graham, C. Kesselman, T. Maquire, T. Sandholm, D. Snelling, and P. Vanderbilt. "Open Grid Services Infrastructure (OGSI), Version 1.0", 27 June 2003, Global Grid Forum, GFD-R-P.15.

[17] Foster, H. Kishimoto, A. Savva, D. Berry, A. Djaoui, A. Grimshaw, B. Horn, F. Maciel, F. Siebenlist, R. Subramaniam, J. Treadwell, and J. Von Reich. "The Open Grid Services Architecture, Version 1.0", 29 January 2005, Global Grid Forum, GFD-I.030.