The WSDM of Autonomic Computing: Experiences in Implementing Autonomic Web Services

P. Martin[1], W. Powley[1], K. Wilson[2], W. Tian[1], T. Xu[1] and J. Zebedee[1]

[1]*School of Computing*
Queen's University
Kingston, ON
{martin, wendy, tian, ziqiang, zebedee}@cs.queensu.ca

[2]*CA Inc*
Sugar Hill, New Hampshire
Kirk.Wilson@ca.com

Abstract

There is a strong relationship between proposed frameworks for autonomic computing, such as the IBM Blueprint for Autonomic Computing, and the Web Services Distributed Management (WSDM) standard proposed by OASIS. We examine this relationship through a description of our efforts to implement an autonomic Web service using WSDM. The example autonomic Web service is based on our Autonomic Web Service Environment (AWSE) framework. We explain how WSDM is used to implement the AWSE components and evaluate the results of the exercise. We present the lessons we learned in carrying out the implementation effort and draw general conclusions concerning the relationship between autonomic computing and WSDM.

1. Introduction

Advances in software technologies and practices have enabled developers to create larger, more complex applications to meet the ever increasing user demands. In today's computing environments, these applications are required to integrate seamlessly across heterogeneous platforms and to interact with other complex applications. The unpredictability of how the applications will behave and interact in a widespread, integrated environment poses great difficulties for system testers and managers.

Autonomic computing proposes a solution to software management problems by shifting the responsibility for software management from the human administrator to the software system itself. It is expected that autonomic computing will result in significant improvements in terms of system management, and many initiatives have begun to incorporate autonomic capabilities into software components. For autonomic computing to be successful, however, vendors must agree upon a common standards-based approach [3].

The Web Services Distributed Management (WSDM) standard [12] is a common management centric interface for a Web services-based environment. It has been identified as an important milestone for autonomic computing for several reasons [4]: WSDM has broad industry support; WSDM provides a necessary management interface to a key technology, and WSDM allows system management platforms to exploit the features of service-oriented computing.

We believe that an important next step towards the realization of the autonomic computing vision is a practical investigation of the suitability of standards like WSDM for implementing that vision. This paper presents the results of such an investigation in which we re-implemented a prototype system called *Autonomic Web Services Environment* (*AWSE*) [15] to be compliant with the WSDM standard. The main contributions of the paper are, therefore, a description of our experiences and a summary of the lessons learned in performing the reimplementation.

The remainder of the paper is structured as follows. Section 2 provides background material on autonomic computing and WSDM. Section 3 outlines AWSE and Section 4 describes how WSDM was used to re-implement our AWSE prototype. Section 5 examines our experiences in using WSDM to implement an autonomic computing system and Section 6 summarizes the paper.

2. Background

WSDM

Web Services Distributed Management (WSDM), from the Organization for the Advancement of Structured Information Standards (OASIS), specifies how the manageability of resources is made available to management clients by means of Web services. WSDM is based on the fact that Web services technology provides an open, standard platform on which to base management infrastructure. WSDM, which is built on top of the OASIS WS-ResourceFramework [6] and WS-Notification [8] family of standards, provides a general specification for management *using* Web services, referred to as MUWS [10], and a specific application of MUWS to the management *of* Web services, referred to as MOWS [11]. MUWS defines how *any* resource can use Web services technologies to expose a manageability interface. MOWS addresses how a Web service as a resource can be managed by means of MUWS. In this paper, we focus on the use of MUWS to manage components other than the Web services in a Web services environment. The implementation of MOWS to manage the Web services is left to future work.

WSDM is based on a model-independent conception of Web service management of IT sources. Under WSDM, it does not matter how individual resources are modeled, as long as they expose their manageability capabilities in a standard way. The WSDM standard describes a set of manageability capabilities that may be exposed by resources. Each such capability is a specific set of properties, operations, and events. For example, the manageability capability of Operational Status, which is used to monitor the health of a resource, is defined by an OperationalStatus property, whose values are enumerated in the specification (available, partially available, unavailable or unknown), as well as by an event type by which managers are alerted to operational status changes of the resource. A resource that implements the Operational Status manageability capability announces itself as exposing the OperationalStatus property as defined in WSDM and (optionally) any related events. Other WSDM manageability capabilities suitable for autonomic computing include Metrics (for interfacing metric and statistical data), Configuration (by which resources expose configurable properties and managers can set those properties), and Relationships (by which resources expose their relationships to other resources).

Autonomic Computing

The Autonomic Computing initiative, spawned in 2001 by IBM, is a proposed solution to the growing complexity of managing large computing systems [2]. The vision is to enable complex networked systems to manage themselves without direct human intervention.

An autonomic computing system manages itself in accordance with high level business objectives, as well as policies and rules specified by human administrators. Autonomic computing systems are self-configuring, self-healing, self-optimizing and self-protecting.

An *autonomic manager* provides the management capabilities of a resource, or a set of resources, called *managed elements* [3] A managed element may be any type of resource, hardware (eg. storage units, servers) or software (eg. DBMS, custom application), that is observable and controllable.

An autonomic manager implements one or more intelligent control loops to perform self-management tasks. The feedback loop consists of four components, namely Monitor, Analyze, Plan and Execute, which are sometimes referred to as the MAPE loop. Sensors provide mechanisms to collect information about the current state of an element. Effectors are mechanisms that change the state or configuration of an element. Central to all of the MAPE functions is knowledge about the system such as performance data reflecting past, present and expected performance, system topology, negotiated Service Level Agreements (SLAs), and policies and/or rules governing system behaviour.

Autonomic managers rarely operate in isolation; they cooperate with other managers to maintain overall system performance, thus requiring communication between autonomic managers. If external management capabilities are required for a component, management interfaces to the autonomic manager must be exposed.

3. AWSE

AWSE is an *Autonomic Web Services Environment*, which is capable of self-management to ensure Service Level Agreement (SLA) compliance [15]. AWSE is comprised of many *sites*, each site consisting of a collection of components and resources necessary for hosting Web services provided by an organization. Individual components, HTTP servers, application servers, database servers, and Web service applications, each have one or more associated autonomic managers.

The original AWSE framework consists of a hierarchy of autonomic managers to facilitate overall system management. The higher level managers query lower level managers to acquire current and past performance statistics, consolidate the data from various sources, and use pre-defined policies and SLAs to assist in system-wide management. At the highest level, a *site manager*, also an autonomic manager, monitors the overall performance of the site and provides service provisioning and management of the components, if necessary, to ensure overall system performance. Autonomic managers, therefore, must be able to communicate to share information.

The initial AWSE prototype was a single site consisting of an HTTP server, an application server and a database management system (DBMS) backend server. The deployed Web service retrieves data from the DBMS. The HTTP server and the application server reside on a single machine with the DBMS on a separate machine. The site manager may reside on any node. The structure of the prototype is shown in Figure 1.

We implemented a single overall manager for each component that oversees a specific resource of that component. For example, the autonomic manager for the DBMS controls the DBMS buffer pools and the autonomic manager for the Web service controls the size of the pool of database connections used by the Web service. The autonomic managers implement the standard MAPE loop using a reflective database-oriented framework as described in detail elsewhere [14].

A reflective system maintains a model of self-representation and changes to the self-representation are automatically reflected in the underlying system. In our case, the self-representation embodies the current configuration settings for the managed element, which control the performance of the element.

The rich capabilities of a Database Management System (DBMS) are used for data storage, creation of a knowledge base, and for controlled execution of the logic flow in the system. The system knowledge base includes system topology, performance metrics, component-based and system wide policies, and the expectations, or system goals. Knowledge used by the MAPE loop is stored in a set of database tables that can be accessed internally by the autonomic element, or externally by other autonomic managers via standard interfaces. DBMS techniques such as triggers and stored procedures are used to implement the logic flow of the autonomic manager.

Although an autonomic component may appear to be performing well in isolation, it may not be functioning efficiently in an integrated environment with respect to overall system objectives. An autonomic environment requires control at the system level to achieve system-wide goals. System level control is implemented by the AWSE site manager.

The site manager requires information about the individual components and the nodes on which they reside to make appropriate decisions regarding resource provisioning or load balancing. A single Management Web service provides access to the information about components residing on a site. This Web service provides two management interfaces; the *Performance Interface* and the *Goal Interface*.

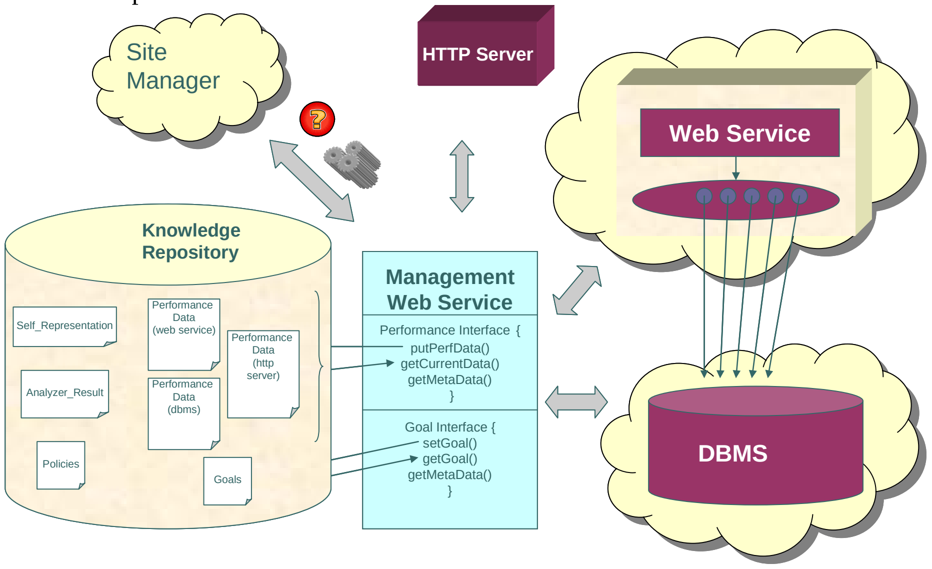

Figure 1: Original AWSE Architecture

The Performance Interface exposes methods to retrieve, query and update performance data for a given component. Meta-data methods allow the discovery of the type of data that is stored for each component. The Goal Interface provides methods to query and establish the goals for an autonomic element, thus allowing external management of a component. Meta-data methods promote the discovery of associated goals.

4. WSDM Augmentation

We used Apache MUSE Version 1.0 [1] to facilitate the reimplementation of AWSE to meet the WSDM (MUWS) standard. The main focus of WSDM is the ***manageable resource,*** which is a resource that exposes its manageability in a standard conformant way. In our AWSE implementation, a manageable resource is a component such as the DBMS, the HTTP server, or a Web service. We therefore had to replace the single management Web service used in AWSE with multiple management Web services, one for each component.

Management information for each manageable resource is accessible through a Web service endpoint, or an *Endpoint-reference* (EPR), as defined in the WS-Addressing standard [13]. The EPR provides a location for the site manager, or other components, to communicate with the manageable resource by means of SOAP messages.

Manageable resources in WSDM are described using XML in a *resource properties document* [7]. This document specifies the *resource properties* that support the manageability capabilities exposed by the managed resource. In AWSE, the self-representation of the system, that is, the current configuration settings for the managed resource, maps directly to the resource properties in WSDM. The component performance goal is also specified as a resource property. MUSE automatically generates the WSDM pre-defined capabilities, whereas component specific manageability capabilities are specified by the developer.

AWSE assumes that individual component performance data is exposed to other components, including the site manager. WSDM provides support for defining performance data using the concept of a *Metrics capability*. The Metrics capability supports metric information relevant to the performance and operation of the resource and allows the specification of metrics associated with each resource. Metrics properties are specified in the resource properties document. For a complex component such as a DBMS, there may be several thousand metrics requiring specification. The metrics capability of WSDM allows specification of the data type and collection interval.

The implementation of the management Web service provides access to the management information for the manageable resource. WSDM specifies a standard manageability interface that allows access to the properties of the manageable resource, and provides external management of the resource. This interface is similar to, and directly replaced, the initial AWSE Management interface as shown in Figure 1. For our prototype implementation of AWSE, the specification of the getResourceProperty() and the setResourceProperty() methods were sufficient for our prototype autonomic managers. Replacing AWSE's common management interface with the WSDM interface for each component required additional programming and increased the amount of required code.

Using MUSE, the developer must implement the back-end code for the Web service methods to do the retrieval or update of the resource property value(s). As this functionality was already implemented in the original AWSE prototype, it was transferred relatively easily to the WSDM version.

Besides querying and configuring resources, using WSDM, the site manager can "subscribe" to receive event notifications when certain changes occur to a resource. Once subscribed, if the value of a property exposed by this capability changes, a notification is sent to the subscriber. In our AWSE implementation, we allow each manageability capability to have an associated WS-Notifications topic, which means that the site manager (or, in fact, any interested component) can subscribe to events regarding this capability by means of this topic. This mechanism allows the site manager to be kept informed of modifications in the system environment and, if necessary, to react accordingly.

The notification mechanism relies on the specifications within the WS-BaseNotification standard [9] and requires that both parties be able to exchange SOAP messages. This requirement meant that the original AWSE site manager, which was implemented as a simple control-panel type interface, had to be re-implemented as a Web service to allow it to subscribe to, and receive notifications from, a component. In addition, in order to generate notification events, it was necessary to re-route the internal call to insert new performance data into the knowledge base through the management web service interface.

5. Evaluation and Discussion

We first evaluate the resulting AWSE-WSDM prototype with respect to three criteria: WSDM's implications for the architecture of the management system, WSDM's support for the MAPE loop, and the amount of complexity introduced by compliance with WSDM. We then discuss the lessons learned in this exercise.

Impact on the System Architecture

An industry-touted advantage of standards is that their interfaces are "implementation independent." This claim, however, is only partially true. Interfaces carry implicit constraints regarding their *possible* implementation architectures. For example, WSDM is designed for a distributed architecture. This "paradigm" application architecture had a significant impact on the AWSE architecture. AWSE originally provided a single Web service access to all components residing on a site. This architecture could not be maintained under the WSDM implementation. We were required to provide separate implementations of the WSDM interface for each component of the system, thereby producing a distributed architecture.

A more subtle impact occurs with respect to our original database-oriented approach. The implementation of notifications forced several architecture and programming changes. In particular, because the MAPE logic was implemented in the database component configuration changes were initiated by updating the component's self-representation in the knowledge base. This update then triggered the actual change to the component. The WSDM interface must be made aware of this change in order to generate a notification. Thus, the call to update the data in the self-representation table had to be re-routed through the WSDM Management Web service.

We see that the paradigmatic architecture on which the WSDM interface specification is based implicitly involves an encapsulated representation of the resource within the resource itself and that the MAPE logic must reside in the manager. The original AWSE database structures must be moved into the software for the best match to the WSDM specification, although such architectural considerations are never specified in the WSDM standard.

Support for MAPE Loop

Considering WSDM's support for the MAPE loop, we found WSDM particularly useful for the interactions among autonomic managers. In our current implementation, WSDM does not, for the most part, affect the internal communication among parts of an autonomic element, or their internal logic flow (although this may change in future work as well).

WSDM provides standard methods for discovering the management interface presented by a management client, which is needed if Web services are to be composed dynamically. We use the management capabilities feature of WSDM to describe the reflective self-representation of a managed element and to expose it to other managers. We found that the WSDM metrics are a convenient way to represent the performance data available from a manager. We also found that the notifications available with WSDM enhance the interactions among managers in AWSE.

It is interesting to note the different role WSDM plays in our reimplementation of AWSE compared to the proposal by Kreger and Studwell [4]. They outline the use of WSDM to provide a manageability interface for a managed element to an autonomic manager. Our work, on the other hand, focused on using WSDM as it pertains to interfaces between managers. The autonomic manager in AWSE presents a manageability interface to allow interaction with other managers. It is expected that in moving to the next generation of AWSE architecture (as described above), we will employ a WSDM interface layer between the managed element (resource) and the autonomous manager.

Amount of Complexity

Considering the the amount of complexity introduced by compliance with WSDM, we found that making our autonomic managers WSDM compliant by using Apache MUSE [1] increased their size significantly. For example, the number of lines of Java code in the implementation of our DBMS autonomic manager was almost doubled. We expect the complexity of both the architecture and the code to increase in the next generation of AWSE.

Our conversion effort, which is described in Section 4, involves five steps. The first step is the creation of WSDL and XSD documents to describe the WSDM Web service and the properties of the managed resource. The second step is stub generation for the Web service using Apache MUSE. The third step is the modification of the stubs to work with our specific resource. The fourth step is the creation of a Backend object to directly manage the resource. The fifth step is the creation of Callback objects to connect each resource property in the WSDM Web service to the Backend object. The approximate numbers of new lines of Java code associated with each of the steps for our DBMS autonomic manager are shown in Table 1. We note that approximately one third of the new lines of code required for WSDM, namely the Java stubs, were generated automatically by Apache MUSE and that these stubs required only a small amount of modification. We also note that, of the remaining roughly 1200 lines of written Java code, only 38% was actually original code. The rest of the written code was taken from provided templates or replicated in multiple locations. We also note that the amount of additional Java code required for WSDM depends upon the number of resource properties being managed and on the number of notification topics used. The DBMS autonomic manager, for example, has ten resource properties and nine custom notification topics.

Step 1: WSDM and XSD documents	270 lines (written)
Step 2: Java stubs	680 lines (generated)
Step 3: Stub modifications	80 lines (written)
Step 4: Backend object	350 lines (written)
Step 5: Callback objects	500 lines (written)

Table 1: Additional lines of code for DBMS autonomic manager

We can also consider the additional complexity involved in WSDM compliance in terms of the number of communication levels involved in processing a client request. In our original version of AWSE, a client request involved four levels of communication. A request is sent from the Web service client to the Apache Tomcat Web container. Tomcat passes the request to Apache Axis, which in turn invokes a method from our Manager Web Service. The Manager Web Service then passes the request to the managed resource for processing. Replies in turn travel back through the four levels.

In the case of our AWSE-WSDM version, a request travels through six levels of communication. A request is sent from a Web service client to the Apache Tomcat web container. Tomcat hands it off to Apache MUSE, which deconstructs the SOAP message and routes the request to the Manager Service through the appropriate Java method call. The Manager Service communicates with the managed resource along a path through the Callback objects and the Backend object. The Backend object is responsible for communicating directly with the managed resource. Replies in turn travel back through the six layers before reaching the client.

We conducted experiments to measure the performance, in terms of requests per minute, of our AWSE-WSDM implementation compared with the original AWSE implementation. The two versions were run with the same workload for fifteen minutes and the overall performance, in terms of requests per minute was calculated. The workload was an online transaction processing workload that involved short query and update requests. The experimental setup consisted of two identical machines (P4 2.8GHZ CPU and 1 GB of RAM) with the database server on one machine and the remaining components plus the workload generator on the other machine.

We saw that the performance of AWSE-WSDM was 7% lower than that of the original version. We believe that the decrease in performance is due primarily to the increased load placed on the application server component of ASWE-WSDM. As explained above, in order to provide notifications, we had to change the structure from the original version such that calls to insert new performance data or to implement changes to the component's configuration are now routed through the WSDM manager. These calls are internal in the original version of AWSE. We did not see any evidence of a significant performance decrease due directly to the extra communication levels imposed by WSDM.

Lessons Learned

Our experiences in converting AWSE to be WSDM-compliant taught us a number of lessons. First, WSDM's standard interface supported our expectations for autonomic computing. The interface allows for easier expansion, which in turn supports scalability. The interface also facilitates dynamic adaptation since it allows an element to discover, and then interact with, previously unknown components.

Second, there is a steep learning curve in order to adopt WSDM, as in learning any new architecture. There are several aspects to WSDM (MOWS and MUWS) and WSDM, in turn, is based on other standards such as WS-Addressing, WS-Notification, and WS-RF. All of the WSDM standards continue to evolve, and we found this added to the difficulty in learning the technology. We also found that this meant that the tools were often behind the standards, and did not support some of the recent features.

Third, WSDM provides a more effective method of communication among components than previously used in AWSE, which employed basic request-response interactions. WSDM also supports the use of subscriptions and notifications. This is well-suited to the interactions between sensors and managers.

Fourth, perhaps the most significant lesson that was learned was coming to appreciate the "paradigmatic" architecture implicit in the WSDM specification. WSDM supports a distributed management architecture. The database-oriented approach used in AWSE is, in retrospect, not a good match to WSDM since the logic flow is modeled using DBMS facilities. We therefore had to make a number of adjustments to AWSE to make it fit with WSDM and these adjustments impacted performance. A more loosely-coupled approach, based on distributed actors, is a more natural fit to WSDM. We plan to investigate this kind of approach in the future.

Fifth, along with the distributed management architecture, we see that WSDM can support different models of control. AWSE supports only a hierarchical model of control where there is a strict hierarchy of managers with the site manager at the root. Our WSDM version of AWSE supports this hierarchical model of control. WSDM can also, however, be used to provide a peer-to-peer model of control where all managers are at the same level and cooperate to arrive at a common management decision.

6. Summary

The success of the autonomic computing paradigm hinges upon the adoption of common standards for the development and deployment of autonomic computing systems. The Web Services Distributed Management (WSDM) standard from OASIS appears to be a standard that is closely linked with much of the functionality envisioned for autonomic computing systems. Practical investigations of the suitability of standards like WSDM for implementing autonomic computing systems are therefore needed to enhance progress in the area. In this paper, we describe our experiences using WSDM to implement a prototype Autonomic Web Service Environment (AWSE).

The use of WSDM added significantly to the amount of code involved in the implementation of our autonomic managers. The majority of this code, however, was either generated by the tools we used, or was code that was replicated in multiple locations. The amount of original code required to accommodate WSDM was not large, nor was it difficult to produce.

Processing client requests involved an additional two levels of communication among components in AWSE-WSDM than in our original implementation. We do not believe that this extra communication significantly increased the system overhead. We found, however, that compliance with WSDM did force us to adopt more indirect logic in some places, which did cause more system overhead.

We also observe that, while standards such as WSDM are meant to be "implementation-independent", they involve implicit assumptions and constraints that dictate aspects of their implementations. In the current implementation, we adapted AWSE in several places where it did not mesh well with this implicit model. Future work will investigate further re-architecting of AWSE to conform to the implicit implementation constraints of the WSDM standard.

7. References

[1] Apache MUSE http://ws.apache.org/muse/.

[2] Ganek, A.G., Corbi, T.A.: The Dawning of the Autonomic Computing Era, *IBM System Journal*, V(42), N(1), (2003).

[3] IBM, "An Architectural Blueprint for Autonomic Computing", June 2005, http://www-03.ibm.com/autonomic/pdfs/AC_Blueprint_White_Paper_4th.pdf.

[4] Kreger, H. and Studwell. T. "Autonomic Computing and Web Services Distributed Management", June 2005, http://www-128.ibm.com/developerworks/autonomic/library/ac-architect/.

[5] OASIS, http://www.oasis-open.org

[6] OASIS, "WS-ResourceFramework". http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsrf.

[7] OASIS, "Web Services Resource Properties 1.2 (WS-ResourceProperties)", http://docs.oasis-open.org/wsrf/wsrf-ws_resource_properties-1.2-spec-os.pdf.

[8] OASIS, "WS-Notification" http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsn.

[9] OASIS, "Web Services Base Notification 1.2 (WS-BaseNotification)", http://docs.oasis-open.org/wsn/wsn-ws_base_notification-1.3-spec-cs-01.pdf.

[10] OASIS, "Web Services Distributed Management: Management using Web Services (MUWS 1.1) Part 1", http://docs.oasis-open.org/wsdm/wsdm-muws1-1.1-os.pdf; and "Web Services Distributed Management: Management using Web Services (MUWS 1.1) Part 2", http://docs.oasis-open.org/wsdm/wsdm-muws2-1.1-os.pdf.

[11] OASIS, "Web Services Distributed Management: Management of Web Services (WSDM-MOWS) 1.1", http://docs.oasis-open.org/wsdm/wsdm-mows-1.1-spec-os.pdf.

[12] OASIS "An Introduction to WSDM", February, 2006, http://www.oasis-open.org/committees/download.php/16998/wsdm-1.0-intro-primer-cd-01.doc.

[13] OASIS, "Web services Addressing (WS-Addressing)", http://www.w3.org/TR/ws-addr-core.

[14] Powley, W., & Martin, P., "A Reflective Database-Oriented Framework for Autonomic Managers". In *Proceedings of International Conference on Autonomic Systems (ICAS'06)*, San Jose, CA, USA, July 16-19, 2006.

[15] Tian, W., Zulkernine, F., Zebedee, J., Powley, W. and Martin, P. "An Architecture for an Autonomic Web Services Environment". *Proceedings of the Joint Workshop on Web Services and Model-Driven Enterprise Information Systems WSMDEIS (ICEIS 2005)*, May 2005, Miami, Fl. pp. 54-66.