A Network Centric Infrastructure for Decision Support Based on Service Integration

J. Berger, A. Boukhtouta, A.-C. Boury-Brisset
Defence R&D Canada – Valcartier
2459, Pie-XI Blvd North
Québec, Quebec
G3J 1X5, CANADA

M. Debbabi, H. Issa, S. Ray, S. Dalouche
Concordia Institute for Information Systems Engineering,
Concordia University,
Montreal, Quebec, Canada

ABSTRACT

The importance of- fresh data is of paramount importance especially in missions involving safety/security critical context. The present infrastructure for service integration implements a one-time execution of a multi-task business process. Business process languages compose various services but the end-result corresponds to a static instance of the retrieved information. This modus operandi exhibits a crucial drawback. Actually, if a Decision Support System (DSS) is to be implemented on top of such integration infrastructure there will be no guarantee of data freshness. Current drawbacks and limitations can be leveraged through a rescue mission scenario, for example, which heavily depends on accurate data. This is true since state-of-the-art technologies associated with web services do not tackle the significant issues of monitoring and re-execution of web services during their integration. Baring this problem in mind, we propose a self-contained middleware capable of performing real-time integration, monitoring and re-execution of web services. Consequently, the end result will guarantee fresh information that is well-suited to impose decision support mechanisms. In this context, this paper leverages the following contributions: First, it identifies a business case for the use of service integration as an infrastructure for distributed decision support. Second, it identifies the software capabilities needed to develop this middleware. Third, it demonstrates a procedure to upgrade the current status of the "Digital Cockpit" middleware [2] to include real-time integration, and monitoring of web services. Finally, it concludes with a partial standard based implementation of the system followed by the snapshots of user interface.

Keywords

Decision Support Systems, Digital Cockpit, Middleware, Monitoring, Re-execution, Service Integration, Web Services.

1. INTRODUCTION

Nowadays, most organizations use a large variety of networked computer systems to collect, process, and produce large volumes of information. In particular, military organizations move towards network-based defense. However, current command and control information systems are highly centralized, using a central processing of information and message exchange between different actors or decision makers. To enable each decision maker or actor to obtain customized and detailed information about the current situation, a command and control system should be decentralized and more dynamic. So, there is a need to provide military decision makers with better decision support systems that facilitate informed and timely decisions in highly dynamic and distributed environments.

The digital cockpit is a middleware that is based on peer-to-peer technology to dynamically distribute situation information between different units or decision makers. Providing the military decision makers with a peer-to-peer information system (or middleware) enables them to have a shared and better understanding of the situation. A peer-to-peer network based defence middleware also enables the decision-making process to be more rapid than previous operational paradigms [32]. Indeed, the ability to dynamically restructure itself and to be able to share information efficiently will be important characteristics of a military force.

The problem of non-fresh information rises due to the static use of information and/or services. This, in turn, does not provide organizations with appropriate access to information in mission-critical situations, where accurate and up-to-date data is aught to be available at all times. In military scenarios, accurate decisions should be taken in an opportune time else the payoff will not be optimal. Achieving a solution for this problem requires real-time or near real-time reaction to events. This is where our proposed middleware fits to close the existing gap by allowing organizations to accomplish high payoffs by utilizing correct information. A decision for a rescue mission planning, for example, is based on a combination of multiple information, such as weather conditions, available aircraft, available troops, etc. As a result, a decision can be made only after all the needed services are composed and executed returning the required information set. Clearly, the main drawback in such a service integration mechanism is that it does not guarantee the accuracy and freshness of data during the whole span of rescue schedule. At any time, information regarding weather conditions may significantly change right after the execution of its corresponding service terminates. Such change is rendered unnoticed and may have significant impacts on the entire previous decision. As a consequence, the entire mission can be jeopardized. This occurs since with current implementations of service integration/monitoring, the decision maker will not be informed about such changes unless s/he requests the re-execution of the entire service process again. Therefore, instead of spending time to analyze data, users will waste more time trying to collect precise data.
Limitations of current service integration frameworks are inherited from the specification guiding web service composition and execution [10], [13]. Identifying that such drawbacks can pose significant problems, our proposed framework builds on top of our previous digital cockpit infrastructure [2], to provide efficient means to overcome current limitations. The enhancement is achieved by adding the capability of performing real-time integration, monitoring and re-execution of web services. By accomplishing this, decision makers can trust the freshness of the delivered information, thus aiding in better decisions. Our proposed infrastructure includes an execution engine, which when detects a change, re-executes the global plan, and returns the

latest information set to the user without any intervention from the client side. This will reduce the time needed for the user to interact with the system, and will introduce more time to accurately analyze the information. The end result will produce a rich blend between the research fields of service integration and decision support systems yielding better systems. The remainder of this paper is organized as follows: Section 2 illustrates the state of the art in both Service Oriented Architecture (SOA) and decision support systems. Section 3 further details our approach. It provides an overview of the digital cockpit's architecture and illustrates how we support web services by adding an integration container. Section 4 identifies a partial implementation of the proposed middleware. We next provide a comparison among various existing technologies that could be used to implement such system, and we present the technologies we used to achieve our goal. We also provide some snapshots of the user interface identifying various levels of information drilling. Finally, section 5 provides our conclusions concerning the entire framework and poses some further research possibilities.

2. SERVICES ORIENTED ARCHITECTURES AND DECISION SUPPORT SYSTEMS

Organizational needs for information integration differ depending on business and operational requirements. Among the most dominant approaches we find: Information Integration, Business Process Integration, and Service Integration [14], [24]. On one hand, information integration is concerned only with data; it allows the combination of data from disparate data sources. With this approach where databases are the main points of connection [3], [17], two main solutions have been proposed: data warehousing and federated architecture. The first requires the building of a physically separated database and the use of Extract, Transform and Load tools (ETL). This solution is relevant when data does not change frequently or when the integrated view does not need to be up-to-date. The federated solution defines one or more mediated schemas, which are not used for storing data but only for querying them. When a query is issued to the system, it is then translated into a set of queries over the corresponding data sources. This approach keeps the local autonomy of data sources and creates a virtual repository enabling real-time on-demand data access. This very solution is appropriate when data or the global schema may change frequently. On the other hand, business process integration approach deals with defining information flow between different systems [8]. Generally, integrated systems are autonomous, exhibit their own process choreography engines, and run internal business processes. The main idea behind business process integration is to provide a single logical model, which spans over multiple applications and data sources.

2.1. Service Oriented Architecture

Organizations are moving towards an information era where significant payoffs could be yielded if business companies implement data/service sharing and integration. The main problem that arises is interoperability. Data sources and services reside on different platforms and each communicates according to its specific communication protocol. Before the emergence of SOA, various integration technologies came to existence. Electronic Data Interchange (EDI) is a standardized message format to perform various business transactions among business partners. Later on, with the advance in communication infrastructure, computing power, and middleware software, new technologies evolved to provide capabilities to integrate data and services belonging to the same application but whose components reside on a distributed computing environment. Common Object Request Broker (CORBA) [25], [28] from Object Management

Group (OMG), Distributed Component Object Model (DCOM) [11], [18], [23] from Microsoft, and Remote Method Invocation (RMI) [27], [26] from IBM and Sun Microsystems are the offspring of these technological advancements. Despite these breakthroughs, these technologies still lack the main essence they were created to solve. Interoperability among different system components residing on different platforms remains weak and difficult to achieve. For example, CORBA and DCOM cannot communicate unless a bridge is placed between them. Still, each of the newly emerged technologies has its own communication protocol. For example, CORBA uses Internet Inter-ORB protocol (IIOP), DCOM utilizes Object Remote Procedure Call (ORPC) protocol and RMI depends on Java Remote Message Protocol (JRMP). Understanding the limitations of existing solutions and to cope with the explosion of the web and e-commerce, SOA for web services came to existence. The concept was created to address the vast integration issues that were prominent from earlier technologies. The new wave of integration that SOA was built on focuses on the independence of platform, languages, data, and communication protocols. Key SOA concepts are based on service description, publication, and binding. Organizations describe their services and publish them in accessible repositories where other services may also be found. This mechanism is achieved in a transparent way hiding all the complexity from the end user. Most definitions of SOA point to the use of web services in their implementation, which can be achieved using any service-based technology [4]. As an example, e-Speak [19], a software product developed by HP in 1999, was the first web service implementation of SOA. Next, companies started developing frameworks for web service integration, resulting in environments such as J2EE from Java, WebSphere from IBM, and .Net from Microsoft. This evolved towards web services that are self-described, self-contained, and modular units built on top of standard-based protocols, and that enable service integration over multiple platforms. Consequently, web services are gaining huge momentum since they overcome the above stated limitations of previous technologies. The architecture of web services is straightforward. Service providers describe their services using Web Service Definition Language (WSDL) and place the corresponding description in a repository. A service requestor uses Universal Discovery, Description and Integration (UDDI) [20] to find the appropriate web service that fits the profile of the requestor. When a match is identified, the service provider and the service requestor bind together and all further communication is achieved via Simple Object Access Protocol (SOAP). This use of standard-based protocols contributed to the wide adoption of web services. However, using web services in such way does not leverage the business world with the real expected value. Therefore, providing a link between web services and business processes will grant the business a precious added value. Among recent initiatives in this direction, we find Business Process Execution Language for Web Services (BPEL4WS) [13] and Web Service Choreography Interface (WSCI) [31]. BPEL4WS is a business process integration specification standard designed by IBM and Microsoft. It provides a language for the formal specification of business processes and business interaction protocols, extending web services interaction, and providing more support to business transactions. Regarding WSCI, it took the first step towards standardization ensuring the adoption of collaborative business applications. Actually, these languages define an interoperable integration model that should facilitate the integration of intra/inter-organizational processes between businesses and organizations.

2.2. Decision Support Systems

Decision makers, in large organizations or military environments rely on decision support systems (DSS) to aid them in the process of evaluating past and present situation and make

informed decisions. This has become a fact since current enterprises are overwhelmed with tremendous amount of information, and it is beyond the human capabilities to fully manage and understand so many data. This is where a DSS steps in and tries to materialize the "big picture" of an organization by better exploiting the information needed to support business processes and decisions. In the military, there is a need to provide military commanders and staff with a common operational picture to gain shared understanding of the situation when making critical decisions. This picture must reflect accurate and up-to-date information coming from disparate and distributed sources. The major components of a DSS are: data sources, user interfaces, architecture and network, and analysis tools. From an architecture point of view, DSS are usually built on top of data warehouses or federated architecture. Thus, analytical capabilities are based on data retrieved from either a data warehouse or a federated view of multiple data sources. The analysis is only initiated after all appropriate queries are executed returning the requested information. This query based dependency weakens the DSS. Based on the fact that queries were designed to be executed once, it limits their ability to capture and reflect any change occurring in a pre-processed field. This major limitation puts the effectiveness, correctness and trust of DSS at stake. Although this is not what decision makers want, this is what they are really getting. To the best of our knowledge, almost all research work on SOA and DSS is done separately. However, providing a framework that provides capabilities to build DSS over SOA becomes a necessity. In this context and in the scope of this paper, we intent to initiate a new paradigm that allows enhancing decision making by taking advantage of what SOA has to offer. These are, asynchronous communications, integration of remote services enabling real-time notification, and service composition. These characteristics provide decision makers with means to make "fresh decisions" based on disparate and accurate information.

3. APPROACH

In this section, we present the approach used to provide a software platform performing real-time integration/monitoring and re-execution of web services. This constitutes a novel approach to establishing an infrastructure suitable for the further conduction of decision support mechanisms in a distributed environment. The cornerstone idea is to establish a profound synergy between the digital cockpit and real-time web service composition and monitoring. We tackle the information and services integration problem from a network centric perspective, focusing on the improvement of information sharing. Moreover, our proposed system improves situation awareness as its core functionality is implemented by an "Integration Container". Consequently, the developed system will form the basis of an up-to-date decision support middleware, better aiding in consolidate decision. Thus the deployed system will yield more mission effectiveness. Hereafter, we briefly present our digital cockpit system [2].

3.1. Software Capabilities and Digital Cockpits

To achieve the required goal and to reach a coherent coalition between the digital cockpit and our proposed middleware, the system should implement the following capabilities:

1- Service integration/composition,
2- Service monitoring,
3- Real-time interactive display,
4- User subscription mechanism,
5- Service re-execution,
6- Analytical methods,

7- Optimization techniques.

The digital cockpit system accomplishes an efficient and useful decision support system mainly based on real-time information retrieval from heterogeneous data sources. It consists of the following modules: Integrator, subscriber, display manager, monitor, analyzer, and controller. Figure 1 shows the layer-based architecture of the digital cockpit system [1].

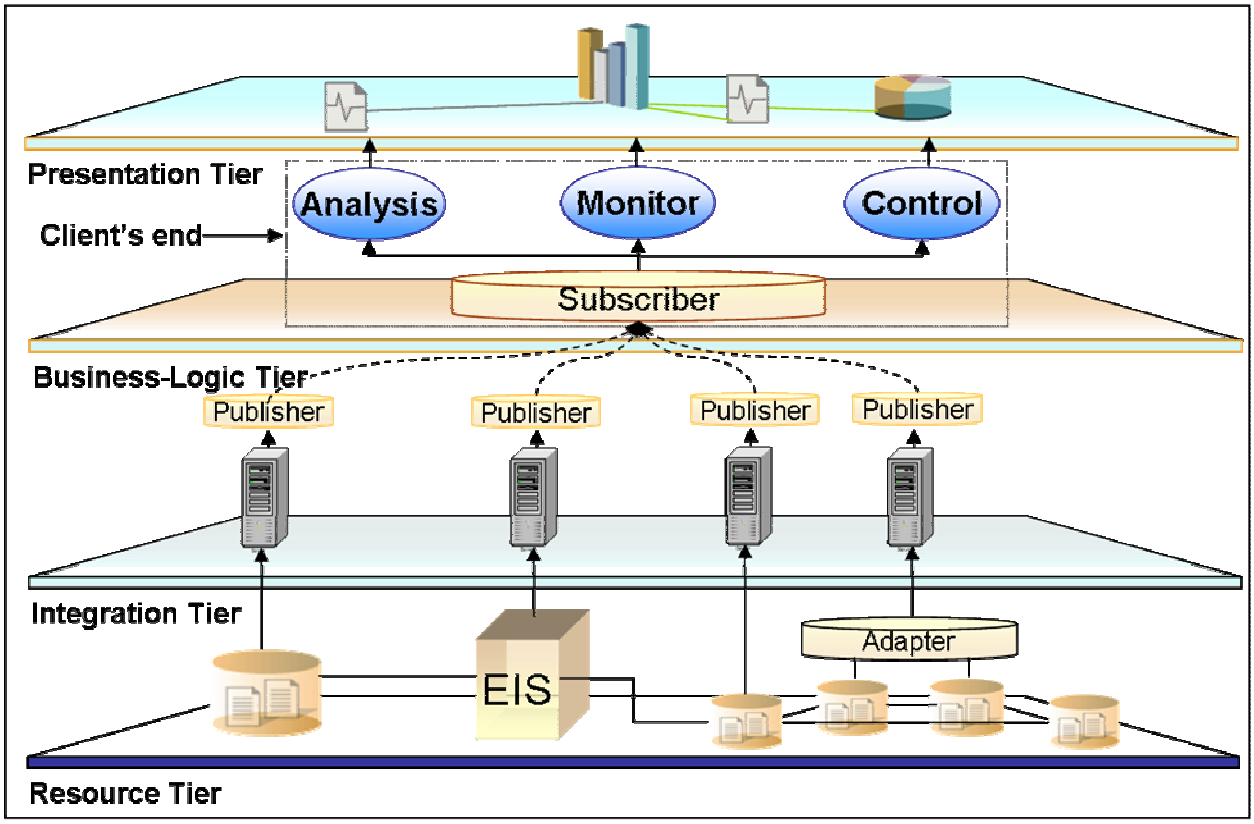

Figure 1: Layer-Based Approach

The resource tier is where all the data needed resides. Data is acted upon and accessed by the integration tier. The main responsibility of this tier is to connect the data sources and integrate them. Following this procedure, the integrator forwards the result to subscribers who initially requested the data, through the acceleration tier. This is achieved by utilizing a publish/subscribe mechanism. At the presentation tier, in the client side, there reside various components that will further act on the data to provide different decision support options. The client side components are called "Analysis" and "Control". While the analysis module is responsible to apply analysis algorithms on the data, the control module is responsible to perform optimizations. All of this are presented to the user in a well structured and sophisticated user interface through various visual components. Each visual component allows different levels of interactions, thus revealing more precise and accurate data with each refinement step. Finally, communication between the components along the hierarchy of the middleware is achieved via asynchronous messages. The digital cockpit middleware provides the following advantages to organizations:

- It offers sophisticated user interfaces that keep on providing the user with up-to-date information without continuous user intervention, using event notification mechanisms.
- Information displayed on the screen is sent by the back-end application, using some middleware technologies (Java Message Service (JMS) in our proposed middleware).
- The use of these advanced technologies leverages many key features such as: guarantee of delivery of information; multicasting of information; and loosely-coupled communication between the digital cockpit client and the back-end application.
- Digital cockpit is intended to provide the decision maker with analytical capabilities, which leverages the decision making procedures.
- Digital cockpit represents multiple software components. Each of them represents a crucial business activity.

3.2. Integration Architecture and Inner Components

The enhanced version of the digital cockpit expands the integration tier to include a novel service integration container. This additional feature rises due to the limitation of the digital cockpit to integrate only heterogeneous data sources and not web services. As a result, adding a new container to the present architecture enables service integration and monitoring. This container will be the core, providing the ability to detect and reflect real-time changes without any user intervention. Within this container resides an execution engine whose duty is to create and manage a business process plan dealing with global service planning, execution, and re-execution. In addition to the execution engine, a notification manager receives updates from corresponding services and feeds these updates to the execution engine. This will automatically trigger the re-execution of the pre-established global plan.

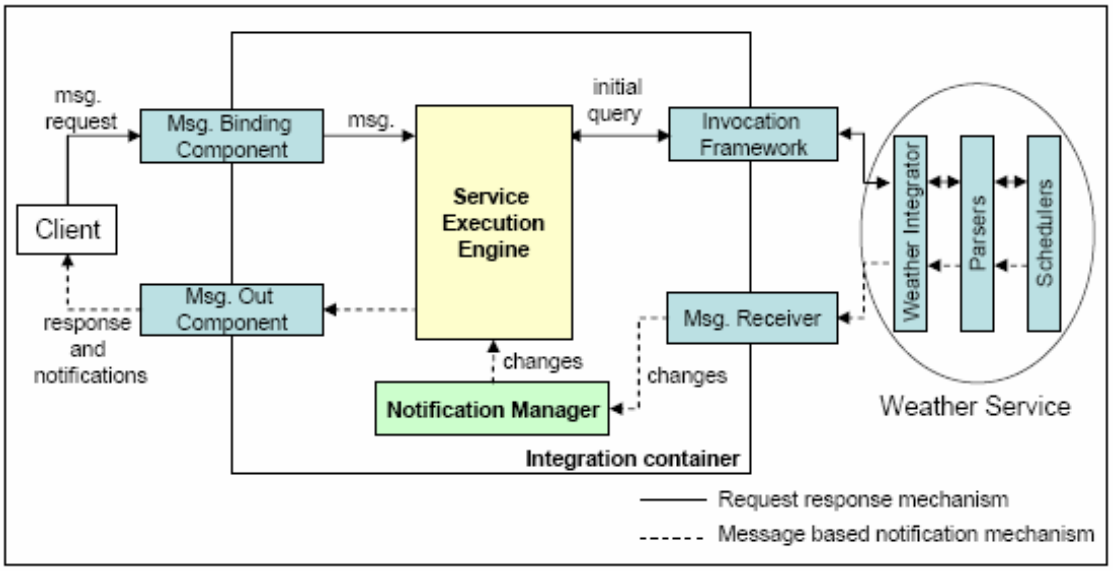

Figure 2: Architecture of Service Integration

Figure 2 represents the overall architecture of the integration container as a standalone system. It exposes the entire process of integration, monitoring and re-execution with a "weather" web service as an example. First, the client sends a request to the integration container indicating an interest with specific web service(s). This request is captured by the binding components and redirected to the service execution engine. Binding components are mediums that transform a request into the BPEL4WS understandable message format and forward messages to the appropriate receiver. The execution engine will in turn compose the global service composition plan and will initiate the execution process. The notification manager will register itself with the corresponding web services. On the other hand, the weather web service has schedulers that inform the notification manager whenever a change takes place in their corresponding datasets. The notification manager will then inform the service execution engine with the changed values. This will re-initiate the process of re-execution. It should be noted that inner/outer interaction between system components and services is achieved asynchronously using messages and publish/subscribe mechanism.

The need of an integration container rises due to the limitations of current technologies used to compose web services [13]. Since the web services are composed and executed in a one-time fashion, the user will not be informed of any change that took place after s/he last submitted a request, unless s/he constantly keeps on interacting with the system requesting it to display its updated results. With the introduction of an integration container, the user interacts only once with the system indicating an interest in some services. From this point, the user is guaranteed to be notified of all instant changes as they occur. This will enable the analytical methods of the DSS to act on more accurate and up-to-date information each time.

1. *Notification Manager:* Once a user identifies an interest in some service(s), the integration container registers its notification manager to all participating services identified during the service planning phase. At this point, we are considering that each service exhibits a mechanism which informs the notification manager that a change occurred within its information set. After receiving this notification, the notification manager will inject this information to the execution manager which in turn will initiate some processes to reflect this update to the user. The importance of this notification manager is that it will free the user from the burden of continuously checking if data has changed since the last request was issued. Moreover, it will enhance the DSS process as a whole because the analytical capabilities of the DSS will process accurate data every time.
2. *Execution Engine:* The execution engine within the integration container is where all the intelligence of this framework resides. Once the integration container receives a request from the user, it will pass it to the execution engine. At this point, the engine initiates a global planning event which decomposes the user's request into smaller ones and builds up business process plan which models the sequence of events that need to be executed. The system will proceed in executing the plan i.e. services and upon completion; the end result will be sent to the user and displayed in a fancy graphical way. At any time, if the system detects a change, the process will re-execute the pre-established execution plan, without any user intervention. This procedure by itself guarantees the freshness of information delivered to the end user.

4. IMPLEMENTATION

The conceptual model of the above-mentioned research is partly validated through an enhanced version of "Digital Cockpit" implementation. In the following section we illustrate the various technologies we used for the development of each component within the system. Technologies are selected after performing a benchmark with other existing technologies, and identifying their strengths and weaknesses.

4.1. Technological Framework

Local communication between classes is achieved by interface contracts, and wiring is achieved using an Inversion of Control (IoC) [6] container called Spring Framework. IoC is an emerging paradigm that greatly improves the reusability, flexibility, maintainability and unit-testability of components. The localization of class instances is transparently achieved through the Singleton and Factory Design Patterns [15] while removing the need for the developer to maintain "Locator" classes. This allows full decoupling of components and system events. There exist numerous Open Source IoC containers available including PicoContainer, Avalon, NanoContainer, Excalibur and HiveMind [12]. However, Spring stands as the best candidate since it has rapidly become the de facto standard for application wiring and lightweight J2EE development. Additionally, Spring Framework provides helper classes that eases J2EE development. Furthermore, Spring integrates nicely with Java Connector Architecture (JCA), Java Database Connectivity (JDBC) and other persistence frameworks to provide declarative local or distributed transaction demarcation.

4.2. Integration Container

Digital cockpit implements a standard integration container of Java Business Integration (JSR 208) [22]. JBI container realizes the access of service in a transparent and independent manner. Therefore, it takes full advantage of the asynchronous features built into BPEL4WS, compared to a classical SOAP over HTTP approach. Since JBI standard is relatively new, there are still few Open Source JBI implementations available, mostly CodeHaus ServiceMix [30] and Sun's reference implementation (RI). The latter provides a more restrictive license than ServiceMix's Apache license, has a few community support, and provides a very limited set of binding components (BC). On the other hand, ServiceMix provides support to Spring as well as JSR 208 deployment unit, it also ships with a number of BC, embeds FiveSights BPEL Process eXecution Engine [9], and is integrated with other CodeHaus project such as ActiveMQ JMS [5], Jencks JCA [29], etc. However, ServiceMix documentation is still in its infancy whereas RI's documentation is a little more complete. Since a BPEL support was needed as well as JMS BC, ServiceMix was chosen. Moreover, developers are responsive and provide good support through mailing-lists and forums.

4.3. Business Process Execution

The execution engine runs business processes and is capable of composing multiple services from distant locations, if required to satisfy a client's request. Incorporating re-computation of business processes is time effective since the process semantics captures the states. The Business Process Execution Language for Web Services (BPEL4WS) provides a high-level language to

compose web services built on web services standards. As a result, our proposed middleware uses BPEL4WS. Among competing BPEL engines, the most popular Open Source alternatives include ActiveBPEL, Apache Twister, Fivesight PXE and IBM BPWS4J. BPWS4J has never been updated and its community support is nearly non-existent. Apache Twister is still infancy and does not provide a complete implementation of the BPEL standard yet. ActiveBPEL engine documentation and community support is excellent, but it provides no other transport mechanism than SOAP over HTTP. As a consequence, ActiveBPEL cannot be easily integrated inside a JBI container. Finally, we proceeded with Fivesight PXE because of its existing integration inside ServiceMix. Due to its extensible architecture, it can be embedded as a Service Engine (SE) inside a JBI container.

4.4. JMS Binding Components

ServiceMix provides support to a number of binding components including ActiveMQ JMS, Jencks JCA, etc. Implementation of asynchronous distributed communication is achieved through ActiveMQ JMS implementation, which is usually considered as the most flexible free JMS broker. Spring framework provides helper classes for JMS templates which avoids programming and maintaining the ever-needed initialization and cleanup code. Furthermore, BPEL4WS supports JMS binding components. However, there is a limitation in the current implementation of Spring JMS template regarding asynchronous reception of messages. This limitation can be circumvented by using Spring JCA support and ActiveMQ JCA Resource Adapter to asynchronously receive JMS messages.

4.5. Service Notification

Integration of services implements initial retrieval of information through an implemented binding component for the invocation of standard web service, called “Web-Service Invocation Framework” (WSIF) [7] developed by Apache foundation. On the other hand the approach encourages a “notification manager” that includes the WS-notification technology to be asynchronously informed from the remote services. However, it should be mentioned that this approach assumes known locations of the remote services. An extension of this approach could be realized to find services dynamically by integrating with an outside XML registry. This issue will not be discussed within the scope of this paper.

4.6. User Interface

The following snapshots of the user interfaces (figure 3 and figure 4) are presented from the enhanced version of the “Digital Cockpit” project, supporting dynamic service integration. It illustrates the integration of a frequently changing weather service. The service interface is written in Digital Weather Markup Language (DWML) format as provided by National Weather Service from USA’s National Oceanic and Atmospheric Administration (NOAA) [21]. This weather service is composed of a set of schedulers and parsers which aggregate weather data coming from various sources and provided under both XML and legacy formats. The JWeather package has been used to parse Meteorological Aviation Routine Weather Report (METAR) legacy format. The Java API for XML Processing (JAXP) has been used to parse XML weather data. In the existence of any change in remote information, this service receives the data in question, by an internal scheduler, and notifies the client through asynchronous JMS notification. It re-calculates the relevant business process, and informs user interface irrespective of any

recursive client's initiative. Finally, the Java3D and JFreeChart libraries have been leveraged for visualizing weather information in a user-friendly way. Java3D provides a high-level programming interface for rendering three dimensional scenes whereas JFreeChart is a widely used Open Source charting library which can generate graphics to be used as textures by Java3D. The following figures represent a "drill down" scenario. A successful integration of weather service in figure 3 shows an overall weather map representing a variety of weather information.

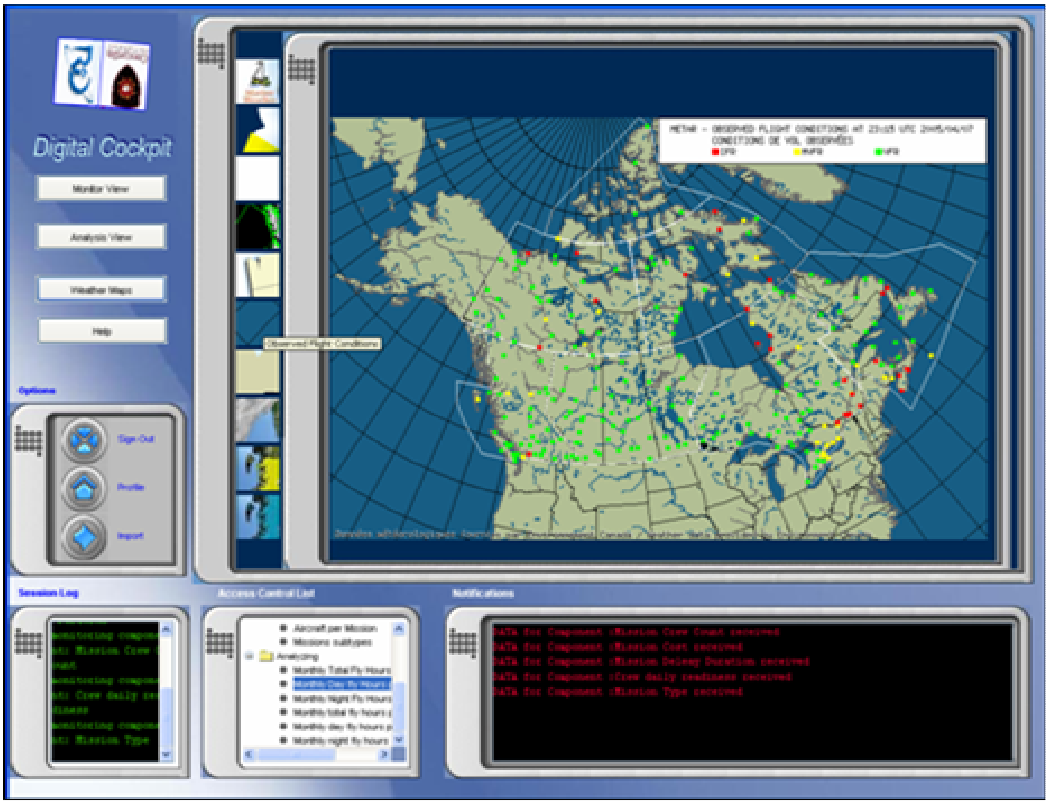

Figure 3: Overall View of Weather Scenario

An interested user may focus on a specific region of interest and requests more information as shown in figure 4. Finally, figure 5 shows a specific wind condition, representing the lowest layer of drilled down information. It should be mentioned that all these visual representations are real-time and update themselves automatically with any fluctuations at corresponding remote ends. At this point, the decision maker is guaranteed to be viewing current and accurate data, and thus will be able to proceed with accurate decision that can maximize the payoff.

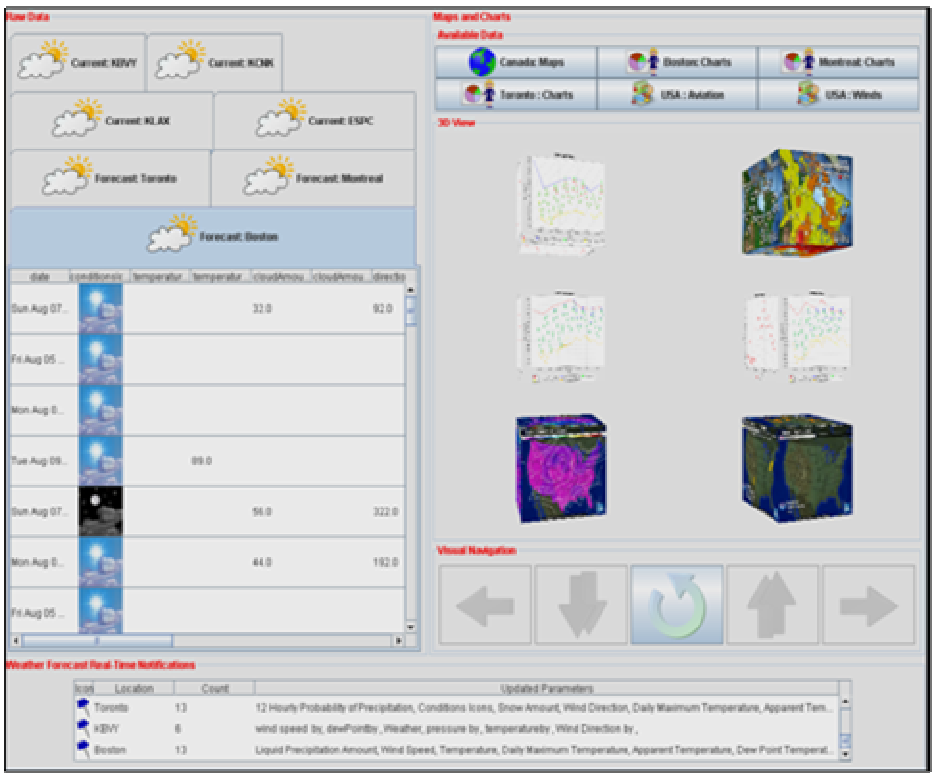

Figure 4: Detailed Information of Weather Scenario

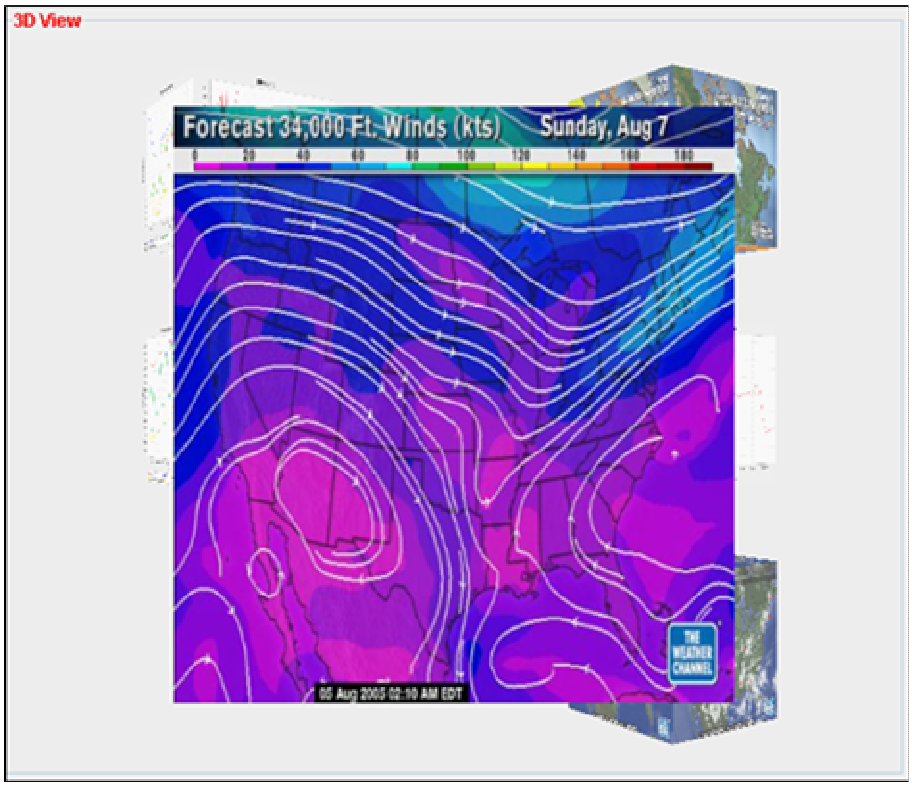

Figure 5: Specific Weather Component - Weather Forecast

5. CONCLUSION

This paper is a result of our extensive research for a decision support middleware platform that envisions real-time, asynchronous integration of data and services. A real-time integration of services is a complicated process due to ad-hoc and proprietary nature of middleware framework and pre-existing remote services. This paper represents a possible standard-based implementation of a well-designed integration architecture that integrates services asynchronously and accepts real-time notification of remote changes during a process run. On the arrival of an event within a business process, the scope of re-computation is identified independently from the integration style. The future work aims towards using such middleware platform for cutting-edge decision support mechanisms that will leverage competitive advantages to organizations. Other research possibilities can focus on the Quality of Service of web service re-execution. This will facilitate faster output by re-executing only a subset of a pre-established business process.

6. REFERENCES

1. A. Benssam, A.Boukhtouta, M.Debbabi, H.Issa, S.Ray, and A.Sahi. A message-based middleware for asynchronous operations: Issues & Experience. In *Proceedings of NOTERE2005*, pages 51–60, Ottawa, Canada, August 2005.
2. A.Boukhtouta, M.Debbabi, and N.Tawbi. A new paradigm for decision making: A synergy between business intelligence and digital cockpits. In *10th ISPE International Conference on Concurrent Engineering: Research and Applications*, Portugal, 2003.
3. dmreview. Guide to Enterprise Information Integration (EII), 2004. http://www.dmreview.com/whitepaper/WID1011596.pdf.
4. Wikipedia encyclopedia. Service-oriented architecture, 2005. http://en.wikipedia.org/wiki/Service-oriented_architecture.
5. Apache Foundation. Activemq 3.1 release, 2005. http://java-source.net/open-source/jms/activemq.
6. Apache Foundation. Inversion of Control, 2005. http://jakarta.apache.org/hivemind/ioc.html.
7. Apache Foundation. WSIF: Web Services Invocation Framework, 2005.http://ws.apache.org/wsif/
8. R. Hackathorn. Convergence: The ethics of business process management, 2005. http://www.bijonline.com/.
9. Quick Hits. Process execution engine (”pxe”), 2005. http://www.fivesight.com/pxe.shtml/.
10. H. Kreger. Web Services Conceptual Architecture (WSCA 1.0), May 2001. http://www-306.ibm.com/software/solutions/webservices/pdf/WSCA.pdf.
11. M. Horstmann and M. Kirtland. Dcom architecture, July 1997. http://msdn.microsoft.com/default.aspx.
12. Java-source.net. Open source inversion of control containers, 2001. http://java-source.net/open-source/containers.
13. M. Juric. Business process execution language for web services: A practical guide to orchestrating web services using bpel4ws, 2005. www.ebpml.org/bpel4ws.htm.
14. D. S. Linthicum. *Next Generation for Application Integration: from Simple Information to Web Services.* Addison Wesley publication, August 2003.
15. J. Maioriello. A survey of common design patterns, 2005. http://www.developer.com/design/article.php/1502691.
16. R. Mason. Implementing an esb using mule: A real world example, 2005. http://www3.java.no/JavaZone/2005/presentasjoner/RossMason/ross_mason-javazone-mule_0.pdf.

17. R. McCann, A. Kramnik, W. Shen, V. Varadarajan, O. Sobulo, and A. Doan. Integrating data from disparate sources: A mass collaboration approach. In *Proceedings of the 21st International Conference on Data Engineering (ICDE 2005)*, University of Illinois, USA, May 2005.
18. Microsoft. COM: Component Object Model Technologies. http:// www.microsoft.com/com/default.mspx, 2005.
19. J. M. Myerson. Web service architectures. Technical report, Sun MicroSystems, 2002. http://www.webservicesarchitect.com/content/articles/webservicesarchitectures.pdf.
20. OASIS. Universal Description Discovery and Integration, 2005. http://UDDI.org.
21. National Oceanic and Atmospheric Administration's. Experimental national digital forecast database xml web service, 2005. http://www.weather.gov/xml/.
22. Java Community Process. Java specification requests (jsr) 208: Java business integration (jbi) 1.0, 2005. http://www.jcp.org/en/jsr/detail?id=208.
23. G. Suresh Raj. A detailed comparison of corba, dcom and java/rmi, 1998. http://my.execpc.com/~gopalan/misc/compare.html.
24. Avanade Software. Enterprise application solution. http://www.avanade.com/_uploaded/pdf/solution/avanadeintegration.pdf, 2005.
25. R. Mark Soley. Middleware that works, 2005. http://www.omg.org/corba-corner/corbalive.pdf.
26. S. Spence. PJRMI:Remote method invocation for persistent systems. In *Proceedings of the International Symposium on Distributed Objects and Applications (DOA'99)*, IEEE Press, Edinburgh, Scotland, February 1999..
27. Sun. Java Remote Method Invocation (Java RMI). http://java.sun.com/products/jdk/rmi/, 2005.
28. S.Vinoski. Corba: Integrating diverse applications within distributed heterogeneous environments. In *IEEE Communication Magazine*, number 2, February 1997. http://www.cs.wustl.edu/schmidt/PDF/vinoski.pdf.
29. Jencks Team. Jencks: A jca container, 2005. http://jencks.org/.
30. ServiceMix Team. Servicemix 1.0 release: An open source enterprise service bus, 2005. http://servicemix.org/ServiceMix+1.0+Release.
31. W3C. Web Service Choreography Interface, 2005. http://www.w3.org/TR/wsci/.
32. T. Gagnes, K. Bråthen, B. J. Hansen, O. M. Mevassvik, and K. Rose, Peer-to-Peer Technology – An Enabler for Command and Control Information Systems in a Network Based Defence? In *Proceedings* of the 9th International Command and Control Research and Technology Symposium, (ICCRTS), Denmark, 2004.