"Service-Oriented Architecture for Smart Vision Application Based Cloud Environment"

Mr. Sachin Darekar
Computer Department
MGM's college of engineering and Technology, Kamothe
Mumbai University
sachindarekar1@gmail.com

Prof. D.R.Ingle
Computer Department
Bharati Vidyapeeth's College of Engineering, CBD Belapur
Mumbai University.
dringleus@yahoo.com

Abstract: Clouds have emerged as a computing infrastructure that enables rapid delivery of computing resources as a utility in a dynamically scalable, virtualized manner. Cloud computing offers its service to the end users on a rental basis which reduces the computing cost of an enterprise or business. Cloud computing is new paradigm for provision of a computing infrastructure and services the over network using a pool of abstracted, virtualized, and scalable, computing resources. One of the challenges is the lack of standard in configuration, management, and programming. Our project aims at creating service oriented application with cloud environment so that we can provide best optimal SOA services which can be deployed on cloud. Distributed application programs have multiple parts that are on different virtual machines. The different nodes can be on the same or different systems.

Keywords - Service Oriented Architecture, Service Oriented Cloud Computing System, Decentralized Software Service (DSS) and Service.

1.0 Introduction

In the recent years, as technology advances and more and more people have their own personal computers, cloud computing has become more popular than ever. Products like Windows 8 using cloud computing and many companies like Amazon and Google are making use of cloud computing. The concept of a uniform architecture for all cloud providers has risen up. Concepts such as service oriented cloud computing have sprung up and have set goals to achieve a uniform architecture that all cloud providers can use so that all people can interact with all cloud providers in a uniform manner. There are many proposed architectures that put forward the ideas on how to make a cloud architecture that will do all this. This paper will take a look at Service Oriented Cloud Computing [1]. The proposed system aspires to improve upon current cloud architectures and make a unified architecture that all clouds should use. This architectures use Service Oriented approach.

Applications lack SOA principles such as reusability, multi-tenancy, flexibility to customize, which can have significant impact on the efficiency of a cloud. We propose a system which will be service oriented where services would be used to build an application.

It enables the user to design applications so that the software modules or components can be loosely coupled; meaning they can be developed independently and make minimal assumptions about their runtime environment and other components. This approach changes how the user can think of programs from the start of the design process and deals with concurrency, failure and isolation in a consistent way.

Description of a system as whole:

In this project we propose a concept to prove CCR and DSS [2], architecture for a Service Oriented application. The proposed system aspires to improve upon the current application building approach and suggests a service oriented approach of creating applications using Microsoft CCR and DSS Introduction Decentralized Software Services (DSS) is a lightweight .NET-based runtime environment that sits on top of the Concurrency and Coordination Runtime (CCR)[3]. Decentralized Software Services (DSS)[4] provides a lightweight, state-oriented service model that combines the notion of representational state transfer (REST)[5] with a system-level approach for building high-performance, scalable applications. In DSS services are exposed as resources which are accessible both programmatically and for UI manipulation. By integrating service isolation, structured state manipulation, event notification, and formal service composition, DSS[6] addresses the need for writing high-performance, observable, loosely coupled applications running on a single node or across the network.

A primary design goal of DSS is to couple performance with simplicity and robustness. This makes DSS particularly suited for creating applications as compositions of services regardless of whether these services are running within the same node or across the network. The result is a flexible yet simple platform for writing a broad set of applications. DSS uses Decentralized Software Services Protocol (DSSP) and HTTP[7] as the foundation for interacting with services. DSSP is a lightweight SOAP[8]-based protocol that provides a clean, symmetric state transfer application model with support for state manipulation and an event model driven by state changes.

The DSS runtime provides a hosting environment with built-in support for service composition, publish/subscribe, lifetime-management, security, monitoring, logging, and much more both within a single node and across the network. Services can be written in Visual Studio, or using Microsoft Visual Programming Language (VPL)[9]. VPL can be used to create applications as compositions of services simply by dragging and dropping them onto a canvas and wiring them together based on their data-dependencies. In addition, DSS Manifest Editor provides a graphical environment for wiring-up, configuring, deploying, and running DSS applications on a single node or across the network.

1.1 Problem Areas Covered by DSS

In the following we describe three common problem areas of application design and how DSS addresses them:

1) Robustness

In any complex system, a failure in a sub-part of the system has the potential of bringing down the whole system. The reason is that a partial failure can lead to a catastrophic failure if it cannot be properly isolated, detected, and handled. Loose coupling is a design pattern that often is invoked as a way of limiting the impact of partial failures. However, in order to build loosely coupled systems, each component must be isolated from all other components as well as from the underlying runtime environment. Two common ways in which systems fail to isolate components are data isolation and execution isolation. Lack of data isolation can cause the internal state of a service to be corrupt and lack of execution isolation can cause a component to become unresponsive. DSS isolates services in both data and execution. Data isolation is achieved by fully cloning all messages exchanged between services (including system services). The cloning code is generated as part of the compilation and is much faster than full serialization.

2) Composability

The requirement of robustness forces applications to become compositions of loosely coupled components. This raises the additional problem of how to identify, locate, and compose such components into a running application. Most traditional systems define an application as a single process and not as a composition of loosely coupled services working together leaving the task of composition to the application designer. DSS provides both protocol and runtime support for creating, managing, deploying, and running applications that are composed of loosely coupled services: A service is created and wired-up at runtime based on a description of which other services it needs to compose with in order to function properly. This model allows applications at runtime to configure where each service instance is running. Further, the relationships that each service instance has with other service instances is exposed through DSSP [10] making it possible to see which other service instances any particular service instance is dependent on.

3) Observability

A critical aspect of any application is that it is possible to know what it is doing, what state it is in, and how it got to be in that state. In short, without a way to observe a system it is impossible to know whether the system is functioning properly. DSS puts the notion of observability at the core of its application model by defining a service as a resource identified by a URI and with exposed state that change as a result of internal service behaviour and through interaction with other services. However, the notion of observability goes beyond just monitoring an application by inspecting the state of its services. By uniformly exposing all services in terms of their state, DSS provides simple and consistent solutions for dealing with administration, management, security, and service composition based on state manipulation. Further, DSS enables every DSS service to be accessed through a Web browser enabling rich UI as well as easy integration with a variety of common tools and platforms.

1.2 CCR Introduction

Concurrency and Coordination Runtime (CCR) [11] is a managed code library, a Dynamically Linked Library (DLL), accessible from any language targeting the .NET Common Language Runtime (CLR).

The CCR addresses the need of service-oriented applications to manage asynchronous operations, deal with concurrency, exploit parallel hardware and deal with partial failure. It enables the user to design applications so that the software modules or components can be loosely coupled; meaning they can be developed independently and make minimal assumptions about their runtime environment and other components. This approach changes how the user can think of programs from the start of the design process and deals with concurrency, failure and isolation in a consistent way.

1.3 Problem Areas Covered by CCR

Asynchrony - When communicating between loosely coupled software components, like programs running across the network, or User Interface (UI) code communicating with the user input and the **file I/O subsystem**, asynchronous operations enable the code to scale better, be more responsive, and deal with failure across multiple operations. Asynchronous programming however, considerably reduces the readability of user code, since logic is often split between callbacks and the code that originates the operation. In addition, it is an almost impossible task to correctly handle failure across multiple outstanding operations.
Concurrency - Code that needs to better utilize multiple execution resources, must be split into independent logical segments, that can run in parallel, and communicate when necessary to produce results from the combined execution. Often, that logical segment is captured by the thread OS primitive that is nothing more than a long lived iteration. Because of thread performance implications on thread start-up, the thread stays active for long periods of time. This forces a particular pattern. Code is structured as long sequences that use blocking or synchronous calls, and only deals

with one thing at a time. Further, threads assume that the primary communication between them is shared memory, forcing the programmer to use very explicit, error-prone methods to synchronize access to that shared memory.

Coordination and Failure Handling - Coordinating between components is where most of the complexity in large software programs lies. A mismatch of interaction patterns, such as calling methods on objects versus using OS signaling primitives versus using queues plus signaling, leads to unreadable code, where the runtime behavior changes drastically between coordination approaches. More importantly, the error handling approaches are ill-defined and again vary drastically.

VPL Introduction

Microsoft Visual Programming Language (VPL)[12] is an application development environment designed on a graphical dataflow-based programming model. Rather than series of imperative commands sequentially executed, a dataflow program is more like a series of workers on an assembly line, who do their assigned task as the materials arrive. As a result VPL is well suited to programming a variety of concurrent or distributed processing scenarios.

VPL is targeted for beginner programmers with a basic understanding of concepts like variables and logic. However, VPL is not limited to novices. The programming language may appeal to more advanced programmers for rapid prototyping or cod.e development. As a result, VPL [13] may appeal to a wide audience of users including students, enthusiasts/hobbyists, as well as possibly web developers and professional programmers.

2.0 LITERATURE REVIEW

The Cloud computing is now being employed to build a massive platform for many company such as Google, Microsoft, Facebook, Yahoo, Ebay, and many more. Underneath the cloud is the use of large scale clustering technology to link together a massive number of computing resources such as computing nodes and storages with high speed gigabit network. Currently, the multicore technology even helps deliver a very high performance computing system at a very low cost for the cloud computing system. Nevertheless, one of the main obstacles for a broad adoption of cloud computing technology is the lack of standard in system the configuration, management, and programming. Most of the well-known cloud implementation is still rely on a proprietary technology. For example, a cloud can be viewed by programmers through various API such as Amazon EC2 API, Go Grid API [14], Sun Cloud API, Elastic Hosts API. Although many open standard efforts are now underway such as OCCI by OGF, the work is still in a very early stage.

Cloud computing system must find a way to solve the problem of service description and conversion, that is, to convert the user's service demand into infrastructure needs. Cloud service [11] users will be a huge user group, and different consumers have different levels of requirements towards Quos. Therefore, when the cloud system set up, it should consider the multiple Quos needs of different users. Static algorithm which is wildly used currently is suitable for a work process of resources allocation without immediate change. Most existing scheduling algorithms are only suitable for simple processes or the Quos constraints that only have one single object. Facing the multiple Quos constraints that cloud computing required, how to ensure multi-level Quos, how to meet the multi-workflow, we need to find out new solutions. The most important advantage of Cloud computing [15] are low-cost equipment and highly universal function. So the hardware can be easily extended, for the cluster data, it needs to divide the task as much as possible. The way of programming have to make a program could automatically run on different scalable processing nodes. The unusual fault conditions of the low-cost machines are far more than the proprietary hardware platform.

Firstly, the design of cloud system must take full account of the machine anomalies. Secondly, it also needs to consider about the problem that the speed does not match the heterogeneous environment, and this has a great influence on the execution of parallel task. Suppose that lets take example for multitenancy issue. Multitenancy is another property that a service oriented architecture must have. This is what a lot of current cloud computing architectures do not have. The reason multitenancy is important is for efficiency. Terms that need to be known are single tenancy and multitenancy. Single tenancy is when a provider has an application running and only one user is using it at a time. An example of a single tenancy program would be a text editor. A user has an application on their computer and only they can run that application. Multitenancy [16] is when a provider has one instance of a program running on a server and many people connect and use the application instance at the same time. An example of a multitenancy program would be Gmail or hotmail.

There are positives and negatives to both of these options. Most of the advantages to single tenancy programs are when the program is being used on a person's personal machine and not when running on the cloud. The major downside to single tenancy programs on the cloud is that they are less efficient because for each person that uses that application there has to be a new instance and this uses a large amount of resources. The positives of multitenancy are everything is handled for the user by the provider. This includes security of user data, backup of user data, and updates to software and hardware. The downsides are that you have to trust that provider is doing their job. There is never any guarantee that the company providing the service will not go out of business or make a mistake managing the user’s data.

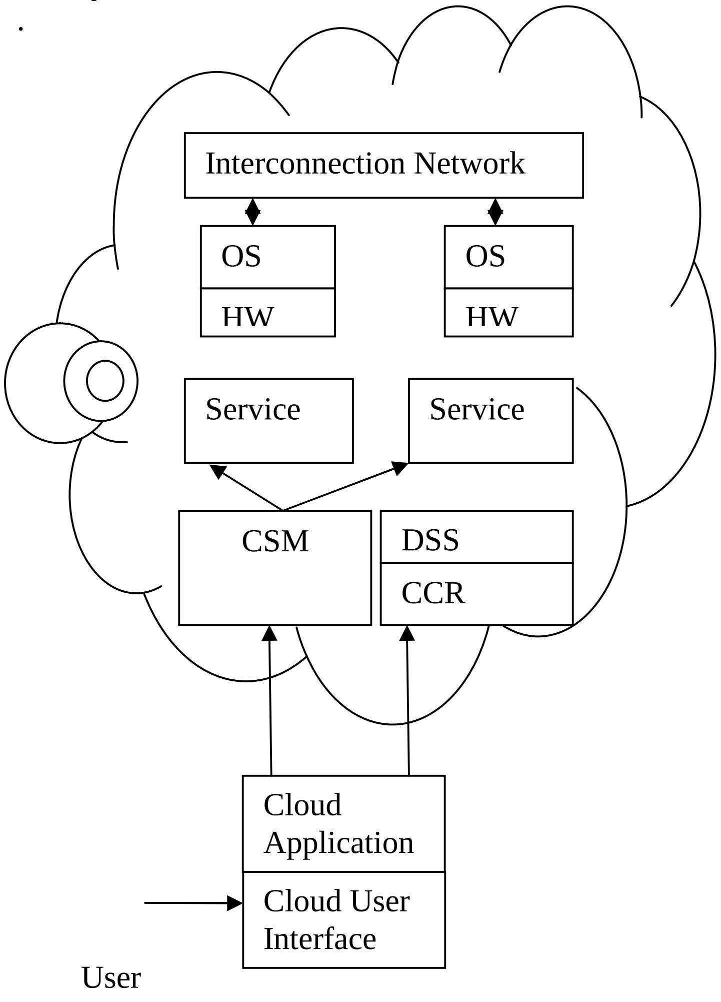
Issues with Current Clouds [17]

- Users are often tied with one cloud provider
- Computing components are tightly coupled
- Lack of Multi-tenancy supports

Thus it becomes very cumbersome and monotonous to follow the approach that was mentioned. As a result, changes can cause confusion as the project team proceeds. The real systems rarely follow the models as proposed. We would appreciate that the proposed system should be more flexible as compared to the former systems.

3.0. System Architecture

.



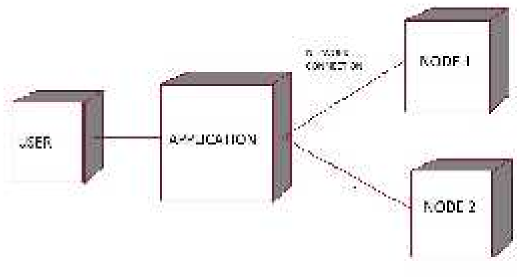


Figure 1: Service Oriented Cloud Computing System

This is the system architecture of our Proposed System which uses service oriented applications in cloud computing which is a method for providing more flexible and scalable service.

In CCR, application is constructed using a set of services. Each service is implemented as a thread pool controlled by Dispatcher. When launched, the main part of the application will break the task to be executed into many small sub-tasks. Then, Arbiter will queue these computing request with data to the work queue of a Dispatcher. The role of Dispatcher is to dispatch the work unit to the threads. After the result has been generated, output is sent to Arbiter again as an event. Then, Arbiter can collect the result and save them to storage for later processing. CCR helps manages both port and threads with optimized dispatcher that efficiently iterate over multiple threads. According to this programming model, a Master/Worker paradigm can be directly apply to structure the parallel and distributed application. Since CCR can manage remote invocation of thread across machine automatically, a single program can easily be scaled to run on multiple machines with a very minor configuration. As the software scale across one machine, the Decentralized Software Services (DSS), a layer of software on top of CCR, can be used to link multiple services component together across multiple machines. DSS provides a lightweight and representational state transfer (REST) oriented application model with a system level approach for building high performance, scalable applications. DSS particularly suited for creating coarse grain applications, and. DSS uses decentralized software services protocol (DSSP) and HTTP for interaction with services, and DSS provides a hosting environment, publish/subscribe, security, monitoring, logging, debugging, they are set of infrastructure services can use for create service.

Fig.1 illustrates the concept behind the software development paradigm used. To developed a cloud application, programmer must decompose the application is to a set of CCR/DSS services and cloud application that integrate and coordinate these services together. The decomposition of service is usually very straight forward in CCR system. An easy to use services development environment (as shown in Fig. 1) that comes with the system can dramatically help shorten the development time. When user application needs resources for the execution, cloud application will send the request to CSM. Then, CSM will discover and allocate a number of services (resources) to users in an on-demand basis. After CSM finally allocates services, it will send services information back to Cloud application. Finally, Cloud application can directly communicate with a group of services and coordinate the execution of services to solve the analysis problem. User can view the results using UI component for user interaction

4.0 Algorithm:

Step1: Start the DSS manifestStep2: Create project
Step3: Select project type
Step4: Adding and configuring nodes
Step 5: Adding and Partnering Services
Step6: Reserve the ports of nodes
Step7: Create Manifests and Deploy Packages
Step8: Run the manifest
Step 10: Stop

5.0 SYSTEM IMPLEMENTATION

System implementation is stage in the project where the theoretical design is turned into the

working system. The most crucial stage is giving the users confidence that the new system will work effectively and efficiently.
The performance of reliability of the system is tested and it gained acceptance. The system was implemented successfully. Implementation is a process that means converting a new system in to operation. Proper implementation is essential to provide a reliable system to meet organization requirements. During the implementation stage a live demo was undertaken and made in front of end-users.

5.1 Methods and Techniques to be used
Here we make use of two different services to create smart vision.
1. Simple Vision
2. Webcam
1. The **Simple Vision** service shows you how to write a service that implements image processing functions using a webcam. This service performs a colour object, a simplified face and hand gestures detections. Other services can get the detection results by subscribing to a Simple Vision service.

2. The Webcam service contract enables you to obtain data from a conventional Webcam (web camera) that is connected using USB or IEEE 1394 (Firewire).

Step 1: Create a Distributed Application Project

Start the Microsoft DSS Manifest Editor and choose **New** from the **File** menu to create a new project.

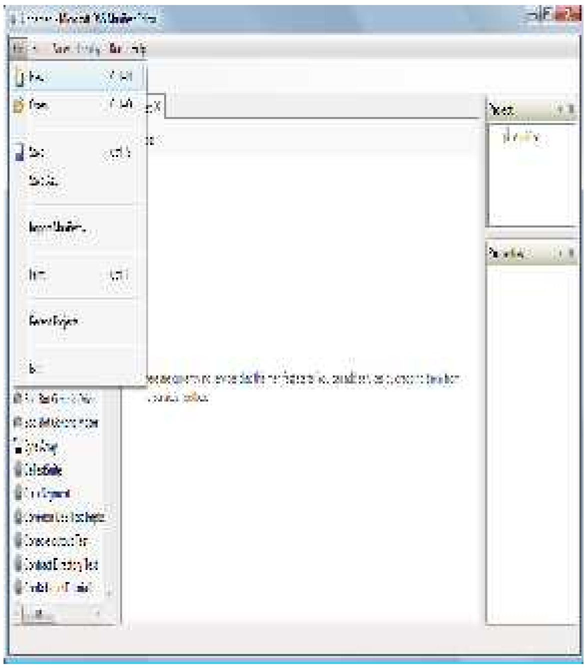
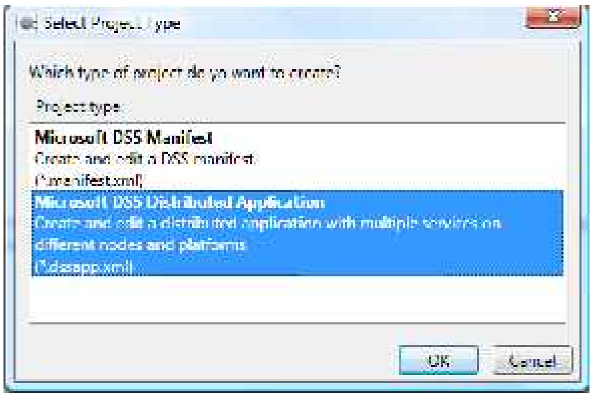


Figure 2 – Creating new project

You can edit a **Microsoft DSS Manifest** for a single node directly. In order to create a distributed application select **Microsoft DSS Distributed Application**.


Select Project Type

Which type of project do you want to create?

Project type

Microsoft DSS Manifest
Create and edit a DSS manifest
(*.manifest.xml)

Microsoft DSS Distributed Application
Create and edit a distributed application with multiple services on different nodes and platforms
(*.dssapp.xml)

Figure 3 – Selecting project type

The **Application** page shows the list of all services in your application independent of the node they run on. The **Nodes** page shows all DSS nodes and the services that they run.

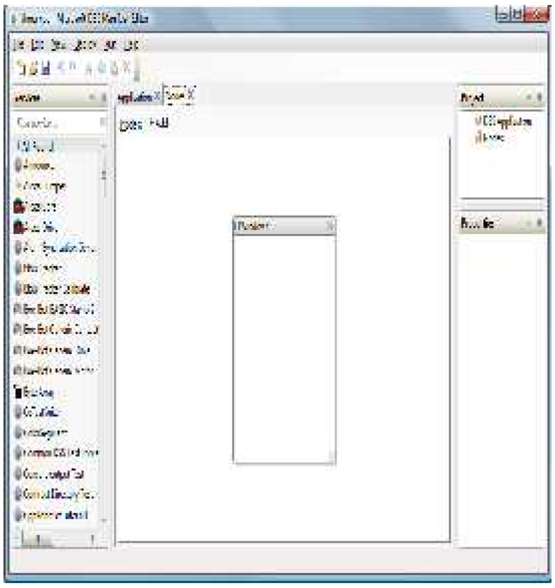


Figure 4 - A new project

Step 2: Adding and Configuring Nodes

Add a second node using the **Add** button (or the right-click context menu).

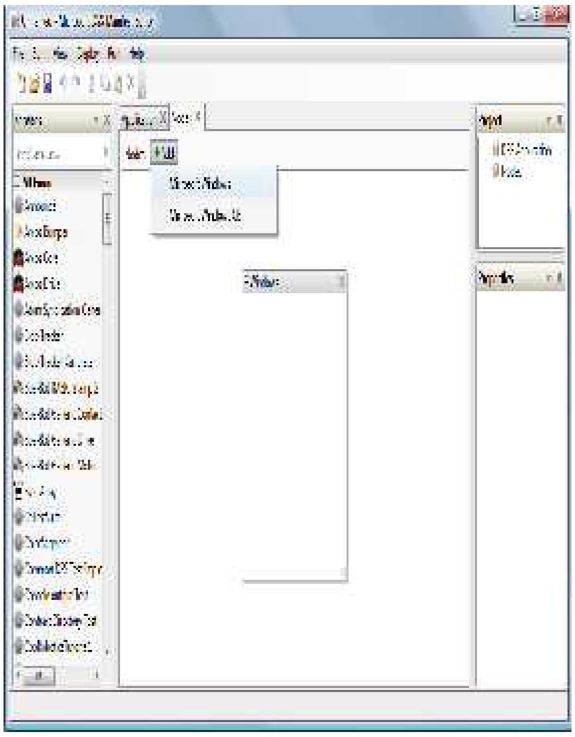
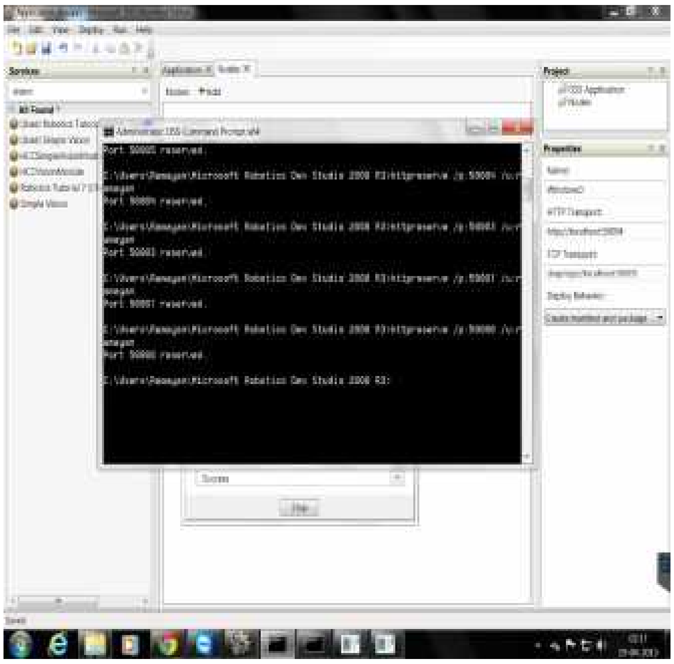


Figure 5 - Adding a new node

The application now has two nodes. Name and allocate the host port address for each node that is to be created. We can easily change this by replacing **local host** in the transport configuration of your node to the DNS names or IP addresses of the computers you want to use. Ports for the HTTP transport must be reserved by an administrator before they can be used



```
Port 50005 reserved.

C:\Users\Ramagan\Microsoft Robotics Dev Studio 2008 R3\httpreserve /p:50004 /u:r
amagan
Port 50004 reserved.

C:\Users\Ramagan\Microsoft Robotics Dev Studio 2008 R3\httpreserve /p:50003 /u:r
amagan
Port 50003 reserved.

C:\Users\Ramagan\Microsoft Robotics Dev Studio 2008 R3\httpreserve /p:50001 /u:r
amagan
Port 50001 reserved.

C:\Users\Ramagan\Microsoft Robotics Dev Studio 2008 R3\httpreserve /p:50000 /u:r
amagan
Port 50000 reserved.

C:\Users\Ramagan\Microsoft Robotics Dev Studio 2008 R3:
```

Figure 6: Adding and Partnering Services

Here the instances of service are added and linked together across different nodes**.** Distribute services across nodes.

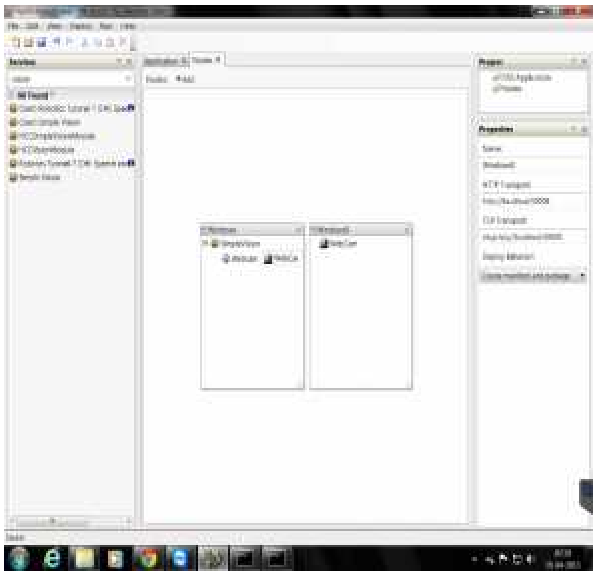
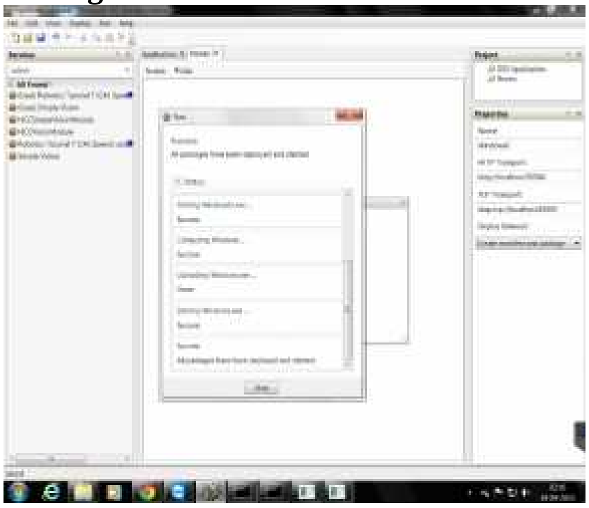
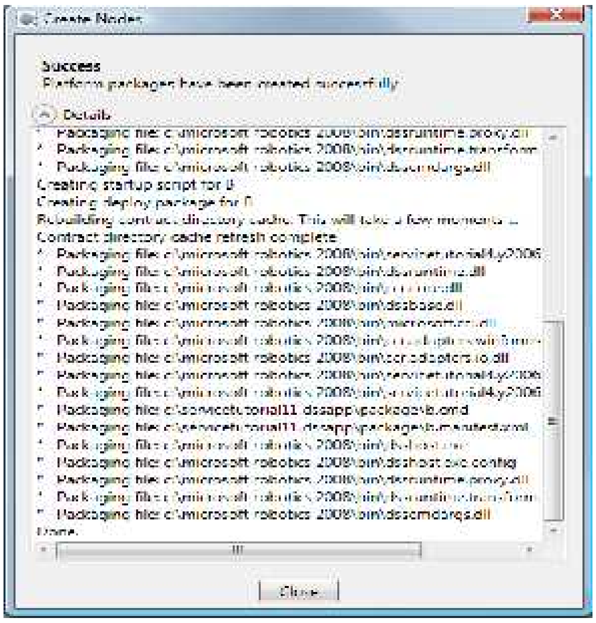


Figure 7: Create Manifests and Deploy Packages

In order to be able to run the application we need to create DSS manifests and DSS deploy packages for each node. From the **Deploy** menu select **Create Deployment Package**.



The deployment packages have been created successfully

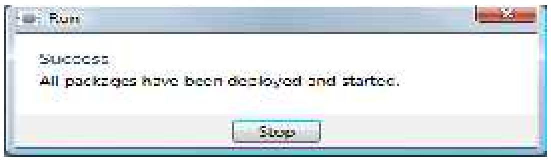


```
Create Nodes

Success
Platform packages have been created successfully

Details
 * Packaging file: c:\microsoft robotics 2008\bin\dssruntime.proxy.dll
 * Packaging file: c:\microsoft robotics 2008\bin\dssruntime.transform
 * Packaging file: c:\microsoft robotics 2008\bin\dsscmdargs.dll
Creating startup script for B
Creating deploy package for B
Rebuilding contract directory cache. This will take a few moments ...
Contract directory cache refresh complete
 * Packaging file: c:\microsoft robotics 2008\bin\servicetutorial4.y2006
 * Packaging file: c:\microsoft robotics 2008\bin\dssruntime.dll
 * Packaging file: c:\microsoft robotics 2008\bin\[illegible].dll
 * Packaging file: c:\microsoft robotics 2008\bin\dssbase.dll
 * Packaging file: c:\microsoft robotics 2008\bin\microsoft.[illegible].dll
 * Packaging file: c:\microsoft robotics 2008\bin\[illegible]
 * Packaging file: c:\microsoft robotics 2008\bin\ccr.adapters.io.dll
 * Packaging file: c:\microsoft robotics 2008\bin\servicetutorial4.y2006
 * Packaging file: c:\microsoft robotics 2008\bin\[illegible]tutorial4.y2006
 * Packaging file: c:\servicetutorial11 dssapp\package\b.cmd
 * Packaging file: c:\servicetutorial11 dssapp\package\b.manifest.xml
 * Packaging file: c:\microsoft robotics 2008\bin\[illegible]
 * Packaging file: c:\microsoft robotics 2008\bin\dsshost.exe.config
 * Packaging file: c:\microsoft robotics 2008\bin\dssruntime.proxy.dll
 * Packaging file: c:\microsoft robotics 2008\bin\[illegible].transform
 * Packaging file: c:\microsoft robotics 2008\bin\dsscmdargs.dll
Done.

Close
```

Figure 8: package deployment

Run the package from a command line or via the Manifest Editor.

This will prompt you to make sure that the **Package Deployer** service is running on each of the computers on which you want to run the nodes .Make sure that the package deployer service is running on the target machines



```
Run

Success
All packages have been deployed and started.

Stop
```

The application is now running. You can inspect the nodes using a web-browser.

Click **Stop** to stop the application and terminate the nodes.

5.2 TECHNOLOGIES USED

Microsoft® Robotics Developer Studio 4

Microsoft® Robotics Developer Studio 4 (RDS 4) is a Windows-based environment for hobbyist, academic and commercial developers to create robotics applications for a variety of hardware platforms. RDS includes a lightweight REST-style, service-oriented runtime, a set of visual authoring and simulation tools, as well as tutorials and sample code to help get started.

RDS includes the following components:

- CCR - Concurrency and Coordination Runtime
- DSS - Decentralized Software Services
- VPL - Visual Programming Language
- VSE - Visual Simulation Environment

Lightweight REST-style, services-oriented runtime

RDS includes a .NET-based REST-style, services-oriented runtime consisting of two components: Concurrency and Coordination

Runtime (CCR) and Decentralized Software Services (DSS).

Microsoft Visual Programming Language (VPL)

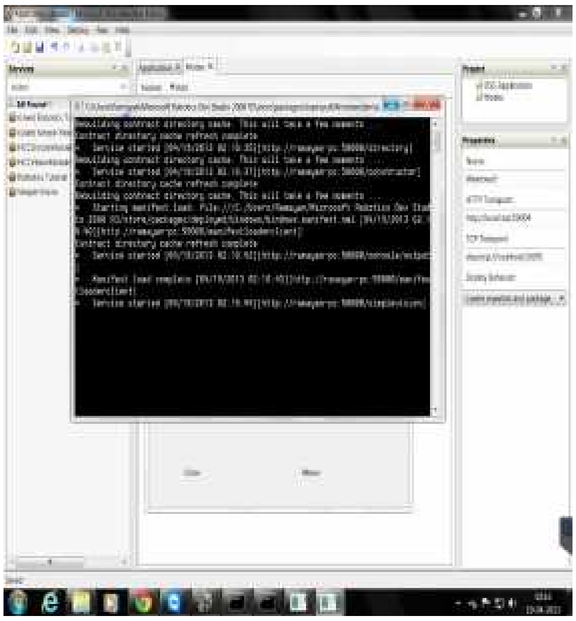
Microsoft Visual Programming Language (VPL) is an application development environment designed on a graphical dataflow-based programming model. Rather than series of imperative commands sequentially executed, a dataflow program is more like a series of workers on an assembly line, who do their assigned task as the materials arrive. As a result VPL is well suited to programming a variety of concurrent or distributed processing scenarios.

VPL is targeted for beginner programmers with a basic understanding of concepts like variables and logic. However, VPL is not limited to novices. The programming language may appeal to more advanced programmers for rapid prototyping or code development. As a result, VPL may appeal to a wide audience of users including students, enthusiasts/hobbyists, as well as possibly web developers and professional programmers

6.0 RESULT AND ANALYSIS

Output with relevant information

The figure 9 shows the implementation of deployment packages.



```
Rebuilding contract directory cache. This will take a few moments
Contract directory cache refresh complete
*   Service started [illegible]
Rebuilding contract directory cache. This will take a few moments ...
*   Service started [illegible]
Contract directory cache refresh complete
Rebuilding contract directory cache. This will take a few moments ...
*   Starting manifest load: [illegible]
[illegible]
[illegible]
Contract directory cache refresh complete
*   Service started [illegible]
*   Manifest load complete [illegible]
[illegible]
*   Service started [illegible]
```

The packages needs to be deployed in order to interface the nodes as shown in fig. 9.

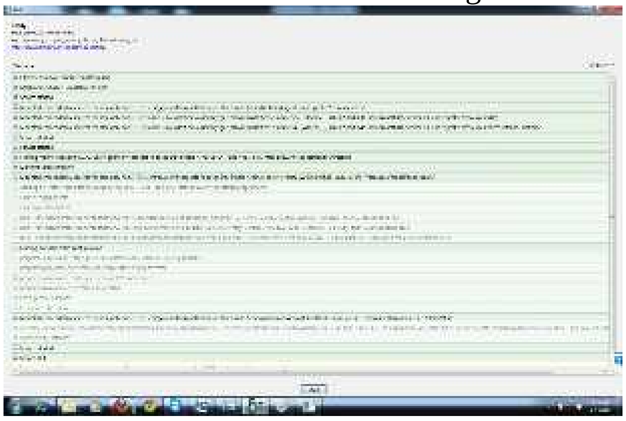


Figure 9: Output with relevant information

The above figure explains all the services that are initiated and run successfully and also determines the error that needs to be addressed.

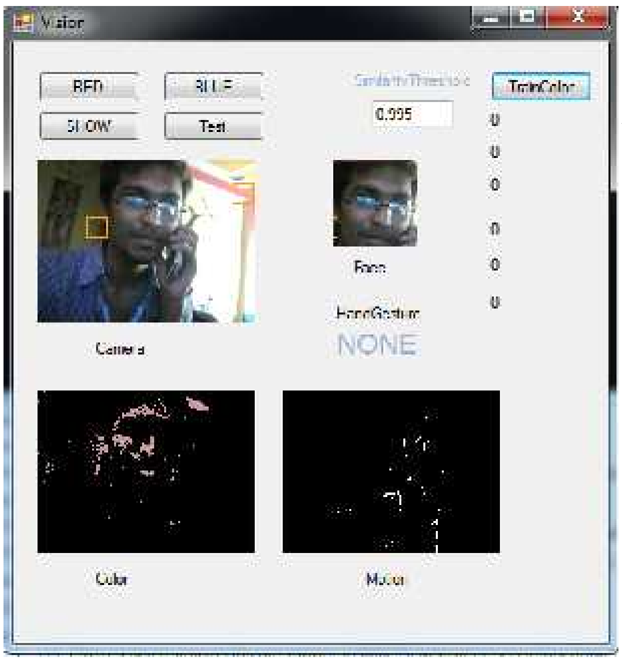


Figure 10: smart vision

Fig 10 shows a DSS node is created and a service instance is generated. While the service is running, it can detect a colour object, a face, etc. The service provides four types of notifications. One for a colour registration, the others, for sending detection results

7.0 Conclusion and Future Scope

Service-oriented and cloud computing combined will indeed begin to challenge the way in which we think about enterprise computing. However, the potential for sharing could not only remove historical barriers but also encourage organizations to think more collaboratively.
In this paper, concluded proposed a service-oriented cloud computing architecture systems (SOCCS) that allows an application to run on different clouds and interoperate with each other. The SOCCA is a 4-layer architecture that supports both SOA and cloud computing. SOCCA supports easy application migration from one cloud to another and service redeployment to different clouds by separating the roles of service logic provider and service hosting/cloud providers. It promotes an open platform on which open standards, ontology are embraced. More scalable and better Cloud Service management (CSM) is planned in future for better interoperability.

8.0 References

[1] Wei-Tek Tsai*, Xin Sun, Janaka Balasooriya Department of Computer Science Arizona State University "Service-Oriented Cloud Computing Architecture" Seventh International Conference on Information Technology 2010.
[2] Rakpong Kaewpuang, Putchong Uthayopas "Building a Service Oriented Cloud Computing Infrastructure using Microsoft CCR/DSS System" Fourth International Conference on Computer Sciences and Convergence Information Technology 2009.
[3] W.-T. Tsai, X. Sun, and J. Balasooriya. "Service oriented cloud computing architecture. Inforamtion Technology, New Generations (ITNG)," Seventh International Conference, pages 684{689, 2010.
[5] L.-J.Zhang and Q. Zhou. Coca: "Cloud computing open architecture". Web Services 2009, ICWS 2009, IEEE International Conference, pages 607{616, 2009.
[6] M.P. Papazoglou et al., "Service-Oriented Computing: A Research Roadmap," Int'l J. Cooperative Information Systems, vol. 17, no. 2, , pp. 223–25 2009
[7] Carl Osipoy, German Goldszmidt, Mary Taylor, and Indrajit Poddar. Develop and Deploy Multi-Tenant Web-delivered Solutions using IBM middleware: Part 2: Approaches for enabling multi-tenancy (2009, May)
[8] Phillip A. Laplante, Penn State University Jia Zhang, Northern Illinois University Jeffrey Voas, SAIC " Distinguishing between SaaS and SOA" 2009.
[9] Liang-Jie Zhang and Qun Zhou CCOA: Cloud Computing Open Architecture IEEE International Conference on Web Services 2009.
[10**]** Ying Huang et al., "A Framework for Building a Low Cost, Scalable and Secured Platform for Web-Delivered Business Services,", 2009.
[11] Wang, Lizhe; von Laszewski, Gregor; Kunze, Marcel; Tao, Jie "Cloud computing: A Perspective study"2009.
[12] Carl Osipoy, German Goldszmidt, Mary Taylor,and Indrajit Poddar. Develop and Deploy Multi-Tenant Web-delivered Solutions using IBM middleware: Part 2: Approaches for enabling multi-tenancy (2009, May).
[13] Ying Huang et al., "A Framework for Building a Low Cost, Scalable and Secured Platform for Web-Delivered Business Services," , 2009.
[14] Phillip A. Laplante, Penn State University,Jia Zhang, Northern Illinois University Jeffrey Voas, SAIC" What's in a Name? Distinguishing between

SaaS and SOA ”P ublished by IEEE Computer Society.2009
[15] Shrikant Mulik, Sushil Ajgaonkar, and Kavindra Sharma, L&T Infotech” Where Do You Want to Go in Your SOA Adoption Journey?” Published by the IEEE Computer Society.2009
[16] Christian Vecchiola, Xingchen Chu, and Rajkumar Buyya, "Aneka: A Software Platform for.NET-based Cloud Computing," in High Speed and Large Scale Scientific Computing, 2010.
[17] B. Kamala, B. Priya, J. M. Nandhini/ International Journal of Engineering Research and Applications (IJERA) “Platform Autonomous Custom Scalable Service using Service Oriented Cloud Computing Architecture” ISSN: 2248-9622 www.ijera.com Vol. 2, Issue 2, , pp.1467-1471, Mar-Apr 2012.