Architecture Level Dependency Analysis of SOA Based System through Π-Adl

Pawan Kumar(1), Ratneshwer(2)

(1) Department of Computer Science (MMV), Banaras Hindu University (Varanasi)
E-mail: pawan.bhuphd@gmail.com
(2) Department of Computer Science (MMV), Banaras Hindu University (Varanasi)
E-mail: ratnesh@bhu.ac.in

ABSTRACT

A formal Architecture Description Language (ADL) provides an effective way to dependency analysis at early stage of development. Π-ADL is an ADL that represents the static and dynamic features of software services. In this paper, we describe an approach of dependency analysis of SOA (Service Oriented Architecture) based system, at architecture level, through Π-ADL. A set of algorithms are also proposed for identification of dependency relationships from a SOA based system. The proposed algorithms would be useful to all stakeholders of SOA based system directly or indirectly. Finally, we automate our approach with a tool developed by us and named 'DA-SOA' (Dependency Analyzer for SOA Based Systems).

Keywords: Architecture Description Language, Dependency Analysis, Service Oriented Architecture, Software Architecture.

1- INTRODUCTION

Software architecture languages (ADLs) provide a means to formally describe software systems at a high level of abstraction. An architecture description should provide a formal specification of the architecture in terms of components and connectors and how they are connected to each other. They capture high level structure and/or behavior of the system, thus provide a basic for coarse grain static analysis [1]. ADLs are intended to play an important role in development of software by composing source modules rather than composing individual statements written in conventional programming languages [2]. Service Oriented Architecture (SOA) is a new form of distributed software architecture. In SOA, the coarse-grained, discoverable, loosely coupled, autonomous services are its basic constitutional units. This makes the SOA different from other architectures for its special architecture elements (services) and its dynamic and evolving structure. An ADL for SOA based system may provide information about interaction among the services, behavior of individual services and prediction of quality attributes. It may also help in future maintenance and comparison of alternative architectural solutions.

In a service based system, since services are loosely coupled together and independently evolved, the change in one service may not detected automatically by other services. Therefore, each change requires a careful analysis of

its impact on other dependent services [3]. The analysis of architectural dependency should be based on depicting the true and holistic complexity of software architecture rather than giving two way relationships among components. If the architecture is complex then it will not be easy to understand and consequent results will appear in designing and coding of the system. Understanding architectural dependencies is about architectural knowledge, not just a question of element decomposition [4]. If a group passes SOA information to their peers by simply handling over unstructured information it can easily become an obstacle for SOA. The information may not be present in the repository, or if it is, it may be too technical for some audiences; unstructured models any need to be produced manually, over and over again, which is time consuming [5].

In this paper, we proposed an approach of dependency identification, at architecture level, for a SOA based system by applying Π-ADL. Recently, a number of ADLs have been proposed such as ACME [6], Rapide [7], Unicon [8] and Wright [9] to support finally representation and reasoning of software architecture. While most ADLs focus on describing software architectures from a structural viewpoint, Π-ADL focuses on formally describing architectures encompassing both the structural and behavioral viewpoints [10]. Various types of dependency relationships for a SOA based system, which may be observed at architecture level, are described. A set of algorithms are also proposed for identification of dependency relationships from a SOA based system. The proposed algorithms would be useful to all stakeholders of SOA based system directly or indirectly. Finally, we automate our approach with a tool developed by us and named 'DA-SOA (Dependency Analyzer for SOA Based Systems'.

The main contributions of the paper are as follows:-

- Description about the various service dependencies that can be observed at architecture level.
- A new methodology to perform dependency analysis of SOA based system at the architecture level using Π-ADL.
- Develop algorithms for identification of the various dependencies
- Design and development of a tool 'DA-SOA' for extracting dependency relationships of a SOA based system.

This rest of the paper is organized as follows. A brief discussion of related work is given in section 2. In section 3, we mentioned the importance of architectural level dependency analysis. In section 4, we briefly describe Π-ADL and explained why Π-ADL is more appropriate for this purpose. In section 5, we described about the various service dependencies that can be observed at architecture level. In section 6, algorithms for identification of the various dependencies are given. In section 7, we describe Design and development of a tool 'DA-SOA' for extracting dependency relationships of a SOA based system. Finally we conclude in section 8.

2- RELATED WORK

In the literature, the problem of software dependence analysis has been studied widely in the context of conventional software, component based systems etc. Substantial work has been reported regarding the dependency analysis. Here, we limit our discussion only to those efforts that are closely related to SOA based systems.

Stafford and Wolf [1], in their paper, have introduced the general challenges of performing dependence analysis on software architecture descriptions, and presented a dependence analysis technique that they have developed for exploring these challenges. They have demonstrated the technique through an example application of a prototype tool named 'Aladdin' that provides automated support for dependency analysis. In their approach they cover conventional software and focused on the direct inter-component dependencies only. Tolksdorf [11], in his paper, proposed a Dependency Mark-up Language to capture dependencies amongst activities and generalization/specialization amongst web services. They have focused on service flow in software processes and used control flow as dependency specification. The resulting specification is more abstract than a concrete control flow and a more specific service description that a functional interface. Ensel and Keller [12], in their paper, presented web based architecture for retrieving and handling dependency among web services with XML, XPath and RDF. Its core component is a dependency query facility allowing the application of queries and filters to dependency models. They focused on dependency representation based on the resource description framework. Basu, Casati and Daniel [13], in their paper, discussed the problem of discovering dynamic dependencies among web services and proposed a solution for the automatic identification of traces of dependent messages, based on the correlation of messages exchanged among services from unstructured message log. This effort is based on analysis of service execution data to discover dynamic dependencies among services. Espinha, Zaidman and Gross [14], in their research paper, demonstrated that how the runtime topology of a SOA basd Syatem can be reverse engineered from observing its operations, and investigated which services are available and how they depend and interact with each other. Their proposed runtime topology reverse engineering approach is implemented in a tool called Serviz. SOA Dependency Analyzer [15] is a tool for graphical visualization of the dependencies between the processes (BPEL-WS) and services (Service Bus). This tool was developed for easy and simple understanding of the dependencies between services and processes, sometimes in very complex environments SOA. It is built on Eclipse RCP (SWT) framework and to visualize the dependency graph used Eclipse GEF/ZEST framework. sCrawler is a dependency tracking utility that tracks dependencies and presents information about them in an application-agnostic manner. It maps the design time dependency of deployed SOA artefacts in an OC4J container [16]. These two tools extract dependency relationships at implementation level. Trigos [17], in his thesis, proposed a model to analyze different approaches for dependency analysis amongst services in a business process. He has developed an algo-

rithm that detects dependencies in the context of their sequence flow and negotiated SLAs. This effort identifies the dependencies at design level.

It can be observed from the above efforts that several works are available to analyze dependencies in a SOA based system at design and implementation level. But very few efforts are available who discussed dependency issue for SOA based system at architecture level. We made an attempt to extend the above contributions further by proposing an approach to identify static and dynamic dependencies, in a SOA based system, at architecture level.

3- ARCHITECTURE LEVEL DEPENDENCY ANALYSIS OF SOA BASED SYSTEMS

SOA based systems are based on the collaboration of reusable services, for which internal source code is not accessible, thus provides limited code visibility at implementation level. Architectural design deals with the quality attributes and structural requirements of the system/elements. Dependency analysis at architectural level, in SOA based system, plays crucial role to develop complex system. Architecture level analysis is being popularized in the current complex software development scenario. There are some major advantages of dependency analysis of SOA based system at architecture level such as early detection of bugs, communication among stakeholders, help in early design decisions, facilitate in maintenance etc.

Modeling architectural dependencies have three important dimensions i.e. type of dependency, source of dependency and degree of effect. Understanding the source of a dependency is an important factor in maintaining a coherent architecture [18]. For architecture level dependency analysis, required information may be text based architecture description, formal language as ADLs and graphical representation as Business Process Modeling Notation (BPMN) and SOA-ML (Service oriented Architecture Modeling Language). Text is the primary means for documenting requirement specification [19]. From the analysis of text based architecture description, existing services and components in SOA based system can be identified. What types of relationship among services exist can be identified. Architecture Level Dependency Analysis using requirement document can be done either manually or automatically. In manually, service specifications are extracted using analyzing the requirement document. In automatically process requirement specifications can be used as an input to the dependency analyzer tool. Such tool scans services/components from requirement documents and shows the dependence relationship among services. Business Process Modeling Notation (BPMN) [20] and SoaML [21] can be used to describe software architecture graphically. To understand the Software Architecture using diagrams is easy for all stakeholders. Conversion from BPMN to Business Process Execution Language and pi- ADL is easy. SOA-ML has different types of diagram which is suitable for modeling of SOA based system. Some diagrams are service interface diagram, service contract diagram, service architecture diagram, service category diagram, etc.

ADLs (Architecture Description Languages) are being popularized for describing software architecture formally. There are a number of ADLs available to describe software architecture formally. Some popular ADLs are ACME [6], Aesop [22], ArTek [23], Darwin [24], MetaH [25], Rapide [26]. Different ADLs have different approaches. ADLs can be used as input of Architecture level dependency analyzer tool. Most of ADLs work for static system. SOA based system is dynamic by nature. Π-ADL is more suitable for describing SOA based system [10]. Since ADLs are formal language in nature, so it has very less ambiguity compared to requirement documents. In Π-ADL, service specifications and their relationship is described formally. So to extract dependency from Π-ADL by scanning services becomes easy.

4- Π-ADL - AN ARCHITECTURE DESCRIPTION LANGUAGE

Architecture Description Languages (ADLs) are being used for decades to describe software architecture. A number of ADLs have been developed and being used to describe the architecture of SOA based systems. In this paper, Π-ADL has been used to describe the architecture of a SOA based systems. We have chosen Π-ADL for this purpose because it describes static and dynamic features of architecture. Π-ADL has been designed under the supervision of Flavio Oquendo in ArchWare European Project in France. It is a formal ADL which is based on mathematical concept 'higher ordered typed π-calculus'. Direct architecture description in Π-calculus of a software system is complex because of mathematical symbol and rigorous mathematical concept. Since architecture description of software system is a formal document which should be understandable by every stakeholder. Majority of stakeholders may not be mathematically sound. For majority of stakeholders to understand architecture description in Π-calculus becomes tedious [10, 27].

Π-ADL is based on π-calculus. Basically architecture can be classified as static architecture, dynamic architecture and mobile architecture. In static system, architecture does not evolve during execution of the system. In dynamic system, architecture can evolve during execution of the system. In mobile system, services can logically move during execution of the system. Π-ADL provides the essential constructs for describing static, dynamic and mobile software architectures. It is a formal specification language designed to be executable and to support automated verification. Π-ADL supports architecture description of a software system from runtime viewpoint. In Π-ADL, software architecture is described in terms of components, connectors, and their composition. General principles which guide the design of Π-ADL are formality, runtime perspective, user-friendliness and executability.

Why Π-ADL is more appropriate for SOA?

To describe the architecture of SOA based system, Π-ADL can be used effectively. Π-ADL has the capability to leverage the benefit of SOA based system. In Architectural term, services are the architectural components, message passing connections are the architectural connectors, and orchestration and choreography the architectural configurations of components and con-

nectors [28]. Services represent computing entities performing a specific behavior within SOA based system and thus they can be specified by means of Π-ADL abstractions. Π-ADL supports for representing dynamic and evolvable architecture. SOA based system is also dynamic and evolvable system. Π-ADL provides a notation to represent the abstraction of structural and dynamic behavior of architecture. It combines predicate logic with temporal logic for supporting the specification of SOA based system. Predicate logic with temporal logic means that truth value of service depends on time. Π-ADL and its tool set may be used for formally developing dynamic web service composition [28]. Π-ADL can be used for modeling the structure and behavior of SOA based system. It supports for formal description of SOA so that verifying quality of SOA based system can be automated. It supports automated verification of structural and behavioral properties of SOA based system. Orchestration and choreography are two ways of composing web services. Composition of web services plays a major role in development of SOA based system. Π-ADL has sufficient constructs to support the composition of services. Π-ADL is able to provide the description SOA artifacts as business processes, orchestrations, choreographies, and interfaces. Π-ADL is service-oriented, formal, practical and executable language.

5- TYPES OF DEPENDENCY FOR SOA BASED SYSTEMS

In order to perform dependency analysis in SOA based system, it is required to identify all dependency relationships. In a SOA based system, the services, the service consumers, and the relevant service stakeholders have to coexist in a system and hence they are dependent upon one other [3].Various types of software dependency are mentioned in literature in context of SOA based systems. We have considered five types of dependency. Web services, in a SOA based system, are dependent upon each other through sending and receiving input and output message. We limit our discussion only for those dependency types that are relevant at architecture level. Some dependencies relevant to SOA based systems have been discussed below. These dependencies are explained with an example of an ‘OnLineBookStore’ website.

5.1 MAIN PROGRAM TO SERVICE DEPENDENCY

Main Program provides interfaces to interact with various services. Many services are called by the ‘Main Program’ to do specific activities. Thus Main Program depends on those services for performing any business functionality. For example, let P is a main program and s1, s2, s3 are services. P calls s1, s2, s3 to perform some functionality, then P is dependent on s1, s2 and s3.

This is shown infigure 1 as follow:

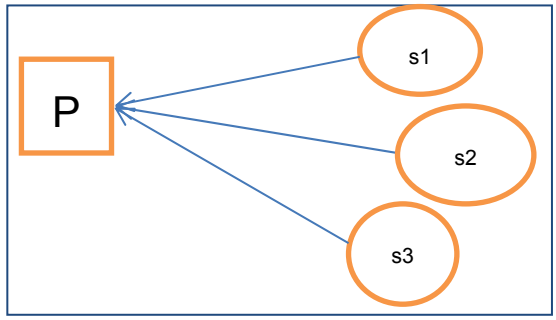

Figure 1. Main Program to Service Dependency

In case of OnLineBookStore website, assume that main program is 'Online-BookStore'. OnLineBookStore uses different services as Search, BookInsert, ShoppingCart etc. Search gives the functionality to main program for searching books based on customer's interest. *BookInsert* provides the functionality to main program for inserting books by user administrator. ShoppingCart takes order from user. Thus the main program is dependent on *Search*, *BookInsert* and ShoppingCartservices for the purpose of searching the books, take order and inserting the books respectively.

5.2 COMPOSITE DEPENDENCY

Composite service is a service which is a combination of different services to perform some functionality. In SOA based system, composition of services is achieved using orchestration and choreography. Suppose s is a service which is composition of services s1, s2 and s3. In this case, s is dependent on services s1, s2 and s3. Service s provides functionality of services s1, s2 and s3. This can be shown in figure 2 as depicted below:

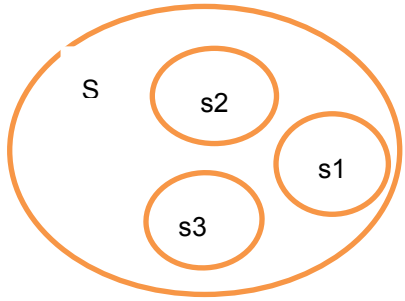

Figure2. Composite Dependency

In case of OnLineBookStore website, assume that there is a composite service: *Search*. *Search* service is composed of SearchByBookName, *Search-ByAuthorName* and *SearchByISBN* services. Thus, search service is depend-

ent on *SearchByBookName*, *SearchByAuthorName,* and *SearchByISBN*.

5.3 CONTROL DEPENDENCY

Control dependence is a situation in which a service's execution depends on another service. A statement in service S2 is control dependent on some service S1 if and only if S2's execution is conditionally guarded by S1. Suppose s1 and s2 are the services and s1 invokes s2 for some function defined in s2 then control goes from service s1 to service s2 in that situation s1 is control dependent on s2 because s1 waits s2 until and unless s2 finishes its execution shown in figure 3.

Figure 3. Control Dependency between s1 and s2.

In case of OnLineBookStore website, *ShoppingCart*, *Order* and *Payment* are three services in which *ShoppingCart* needs *Order* Service for fulfill the customer's order and *Order* Service needs *Payment* Service for books' payment. Service *ShoppingCart* is control dependent on *Order* and *Order* itself is control dependent on *Payment* service where service *Order* needs confirmation about *Payment* so control jumps from *Order* to *Payment* and after confirmation of *Payment* control comes back to service *Order* for further operation.

5.4 DATA DEPENDENCY/MESSAGE DEPENDENCY

Data Dependency comes under consideration when one service calls to another service and passes some data or messages to another service as input parameter. In this case the service-that receives data is data dependent on the service which is sending data. If invoking service gets data from invoked service as output then the invoking service is data dependent on invoked service. Suppose s1 and s2 are the calling and called services respectively. If s1 calls s2 and send some data to s2 as input parameter then in that case s2 is data dependent on s1. And if s2 returns value or data to s1 then s1 is data dependent on s2 because s1 waits for the result from the called service s2. Figure 4 shows Data Dependency by red arrow.

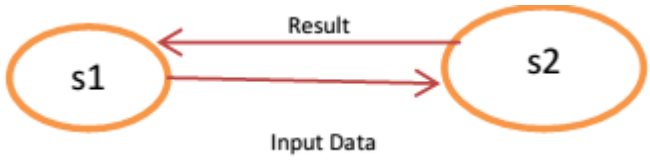

Figure 4. Data Dependency

In case of OnLineBookStore website, ShoppingCart uses Order Service for books's order. *ShoppingCart* is the calling service calls to service *Order* and passes BookInformation as data. In that situation *Order* is data dependent on *ShoppingCart* and *Order* also sends success or failure transaction result to *ShoppingCart* as output. Thus *ShoppingCart* is data dependent on *Order*.

5.5 SEQUENCE DEPENDENCY

Sequence Dependency depicts the flow of service dependency and shows sequence of service execution. Suppose s1, s2, s3 and s4 are the services. s1 depends on s2, s2 depends on s3 and s3 depends on s4 sequentially. Figure 5 shows sequence dependency.

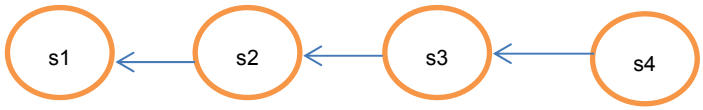

Figure 5. Sequence Dependency

Example system *OnlineBookStore* provides the main interface to the user. When user wants to order any items from the search interface, requires login before order any items and makes payment after order, Thus Order service is sequentially dependent on Login service and Payment service sequentially depends on Order service.

6- PROPOSED ALGORITHMS FOR DEPENDENCY EXTRACTION FROM Π-ADL DOCUMENTS

In this section, we have described the proposed algorithms for different types of dependency (that have been discussed in previous section). These algorithms have been written in pseudo code for better understandability. In each algorithm, assignment symbol is shown as "←" and variables are written in italics, "ϕ" has been used for empty list. In all algorithms basic input is architecture description, written in Π-ADL, of the module project "online bookstore". Keyword used in Π-ADL has been written in 'Arial' font and remaining text in font 'Times New Roman' to differentiate between keywords and other texts written in algorithm. One special symbol " ∈ "has been used to extract element from list one by one in ordered fashion. Algorithms are given in following subsections.

6.1 'MAINPROGRAM TO SERVICE DEPENDENCY' EXTRACTION ALGORITHM

This algorithm extracts dependency information of all services that provide functionality directly to the main program. We have named this algorithm as 'MainProgramtoServiceDependency'. Architecture description of a module project 'online book store' (a SOA based system) using Π-ADL is taken as in-

put. Variable *maintoServicetree* has been taken as empty list i.e. this variable is assigned by ϕ (empty list) .We have used three pattern *ArchitecturePattern*, *ComposePattern* and *ServicePattern* to extract dependency information from Π-ADL as follows:

ArchitecturePattern:

@"\b(architecture|service)\s+[a-zA-Z_]\w*\s+(is)\s+(abstraction\((\w+\s*\w+)?\))"

ComposePatern:

@"(compose)\s*[\{]\s*[a-zA-Z\(\)\.\s*]*[^\}]"

ServicePattern:

@"(via)\s+(?<sName>(\w+))\s+(send\(\))"

In the above pattern the symbol @ denotes that there may be special symbol included in this pattern. \b denotes the ignorance of first consecutive blank spaces. Architecture or service denotes the keywords used before service name or architecture name. \s+ denotes for one or more spaces. [a-zA-Z_] denotes the one letter of lower case or upper case of alphabet or underscore. W * denotes that alphabets or digits may occur zero or more times.'is' and 'abstraction' stands for keyword in Π-ADL program. (\w+\s*\w+)? denotes zero or one time of (\w+\s*\w+).

'Compose' in ComposePattern is a keyword which is used to show the composition of more than one services. 'via' in *ServicePattern*is a keyword which is used for interaction using sending and receiving using the keyword send and receive.

ArchitecturePattern recognizes the following line in Π-ADL:

architectureOnLineBookStore is abstraction()

On the basis of *ArchitecturePattern, the* Π-ADL program is divided into different blocks which are stored into the variable list *ADLProgramPartition*. We have used one variable *ServiceList* which stores all services by calling procedure *GetAllServices( )*. Explanation about the algorithm *GetAllServices* has been given in the last subsection.*part* is a variable which stores the first element of *ADLProgramPartition*. Since here main program is a main architecture which calls other services or architectures. So we have taken *parent* as first element of *ServiceList.* Since first element of ServiceList is main architecture and it calls all the services which are initialized in block *part* as composition of services and architecture. *ComposePattern* has been used to extract the compose block from the block *part*. *cservice* is a variable in which compose block from block *part* has been assigned. *ServicePattern* is used to extract services from compose block*csservice*. *foreach* loop has been used to extract the element from *csservice* and assigned to variable *s*. Inside this loop list*Services* has been assigned by the elements which is extracted from *s* on the basis of ServicePattern.

Algorithm 1: Proposed Algorithm for 'MainProgram to Service Dependency' Extraction

```
1. Algorithm: Extract MainProgram to Service Depend-
   ency from ADL
2. Algorithm:MainProgramtoServiceDependency()
3. Input : Π-ADL
4. Output :The output will list out all services that
   provide functionality to main program i.e. having
   MainProgramtoServiceDependency
5. // Start of the Algorithm
6. {
7. maintoServiceTree← ϕ;//ϕ is used for empty tree
   list
8. // Pattern Matching logic
9. //Architecture Pattern has been used for dividing
   the Π-ADL program into blocks
10.     ArchitecturePattern←
   @"\b(architecture|service)\s+\w+\s+(is)\s+(abstrac
   tion\((\w+\s*\w+)?\))";
11. // Compose Pattern  has been used to extract com-
   posed service block of a service
12. ComposePattern←       @"(compose)\s*[\{]\s*[a-zA-
   Z\(\)\.\s*]*[^\}]";
13. // Service Pattern has been used to extract ser-
   vice name from composed service block
14. Service                  Pattern                ←
   @"(via)\s+(?<sName>(\w+))\s+(send\(\))";
15.     ADLProgramPartition←  Split Π-ADL    on the
   basis of ArchitecturePattern;
16. //Calling of GetAllServices procedure which is
   given   at the end of this section
17. ServiceList←GetAllServices();
18. part←ADLProgramPartition[0];
19. // Logic for main program to service dependency
20. //This extracts composite service from part in
   pi- ADL;
21. cservice←getMatches from part  on the basis of
   ComposePattern;
22. parent←ServiceList[0];
23. //This is the main loop for composite service ex-
   traction
24. foreach(s∈cservice)
25. {
26. Services←getMatches from s on the basis of Ser-
   vicePattern;
27. if (Services.Count>0)
28. {
```

```
29. maintoServiceTree.Nodes←parent;
30. foreach(s ∈Services)
31. {
32. child←s.sName;
33. maintoServiceTree.Nodes[0].Nodes ← child;
34. }
35. }
36. }
37. return maintoServiceTree;
38. } //End of mainprogram to service dependency al-
    gorithm
```

Inside the *foreach* loop in algorithm an *if* statement has been used. If number of elements in Services is more than zero*, then maintoServiceTree.Node* is assigned by parent. Again another *foreach* loop is defined inside *if* statement. This loop is used to extract service *s1* from list *Services* one by one. *child* is a variable which stores the name of a service *s1* by *s1.sName* on the basis of *ServicePattern*. After this *maintoServiceTree.Nodes[0].Nodes* is assigned by *child.* The relationship between main program to services is stored in mainServiceTree. Finally mainServiceTree is returned.

6.2 'COMPOSITE DEPENDENCY' EXTRACTION ALGORITHM

This algorithm extracts composite dependency from architecture description of the module project 'OnLineBookStore' written in Π-ADL. The name of this algorithm is CompositeDependency(). The algorithm takes Π-ADL as input and gives the list of composite dependencies as output and this output is stored in list CompositeDependencyListTree.

We have taken two lists CompositeDependencyListTree and ServiceList. Initially both lists are assigned by ϕ (empty list). CompositeDependencyListTree has been taken for storing all the composite dependency relationship and ServiceList to store all the services used in the module 'OnlineBookStore'. The algorithm Compositedependency uses the pattern matching approach from architecture description to extract the composite dependency exist in module. Pattern used in this algorithm are **ArchitecturePattern** ,**ComposePattern** and **ServicePattern**. The explanation about these patterns has been given in the description of the algorithm MainProgramtoServiceDependency. CompoiteDependency( ) calls the algorithm GetAllServices( ) which returns all services exist in module 'OnLineBookStore'. These services are stored in the list ServiceList.

We have taken ADLProgramPartition as variable of list type. Π-ADL is broken into different blocks on the basis of ArchitecturePattern. These blocks are stored into list ADLProgramPartition. Two counters have been taken as *rootcount* and *j* which are initially assigned by 0. Three *foreach* loops have been used which are nested. In first *foreach* loop a variable *part* has been used to extract elements from the list ADLProgramPartion one by one taking

in order. Under this foreach loop a variable '*a'* has been used to take service from *ServiceList* using the subscript '*j*'. We have taken a *variable list composeservice* in which extractions from *part* have been stored on the basis of ComposePattern. Under first foreach loop, second foreach loop has been declared. In control part of second foreach loop a variable '*cs'* has been used to extract the element from *composeservice.* In the body of second foreach loop a variable list *services* has been declared. '*services'* has been assigned by the elements extracted from 'cs' on the basis of ServicePattern.

If the elements in services is greater than 0 i.e. if(services.Count>0) then *CompositeDependencyListTree.Nodes* is assigned by' *a'.* This 'if' statement has been used in the body of second '*foreach'* loop. In the body part of *if* statement, third *foreach* loop has been defined as '*foreach(s ∈ services)*' . In this loop *s* is a variable which takes element from *services* one by one. *sName* is used for name of service which is defined in the *ServicePattern.* Variable *b* has been used to store the service name by the statement '*s.sName*'.

Algorithm 2: Proposed Algorithm for Composite DependencyExtractionfrom ADL program

```
1. Algorithm: CompositeDependency( )
2. Input: Pi ADL
3. Output:  The output will list out all composite to
   automic or composite service relationship i.e. Com-
   positeDependencyList
4. //start of the Algorithm
5. {
6. CompositeDependencyListTree←ϕ;// at initial Compo-
   siteDependencyList is empty
7. ServiceList←ϕ;                                  //at
   initial ServiceList is empty
8. // Pattern Matching logic
9. //Architecture Pattern has been used for dividing
   the Π-ADL program into blocks
10.      ArchitecturePattern←
   @"\b(architecture|service)\s+\w+\s+(is)\s+(abstract
   ion\((\w+\s*\w+)?\))";
11.  // Compose Pattern  has been used to extract com-
   posed service block of a service
12.  ComposePattern← @"(compose)\s*[\{]\s*[a-zA-
   Z\(\)\.\s*]*[^\}]";
13.      ADLProgramPartition←Split π-ADL Program on
   the basis of ArchitecturePattern;
14.  ServiceList←GetAllServices( );
15.  //Calling of GetAllServices procedure which is
   defined at the end of this section
16.  //Logic for Composite Dependency
17.  rootcount←0;
```

```
18.  // rootcount has been used for subscripting of
     service which is composite service
19.  j← 0;
20.  // j  has been used to for subscripting of all
     services one by one
21.  // using foreach loop member of ADLProgramParti-
     tion is assigned to part one by one
22.  foreach(part ∈ ADLProgramPartition)
23.  {
24.  a←ServiceList[j];
25.  composeservice←getMatches from part  on the ba-
     sis of  ComposePattern;
26.  foreach(cs∈composeservice)
27.  {
28.  services←getMatches from cs on the basis of Ser-
     vicPattern;
29.  if (services.Count>0) then
30.  {
31.  CompositeDependencyListTree.Nodes←a;
32.  foreach(s∈ services)
33.  {
34.  b←s.sName;
35.  CompositeDependen-
     cyListTree.Nodes[rootcount].Nodes←b;
36.  }}
37.  rootcount← rootcount+1;
38.  }
39.  j←j+1;
40.  }
41.  returnCompositeDependencyListTree;
42.  } // end of composite service dependency algo-
     rithm
```

This line in algorithm '*CompositeDependencyListTree.Nodes[rootcount].Nodes← b';* shows that CompositeDependencyListTree stores the atomic service b to the composite service at rootcount location. When all foreach loops terminates then all the composite dependencies among services are stored in CompositeDependencyListTree. After that 'CompositeDependncyTree' is returned by this algorithm.

6.3 'CONTROL DEPENDENCY' EXTRACTION ALGORITHM

This algorithm extracts control dependency from architecture description of module OnLineBookStore (SOA based system) which is described in Π-ADL. We have named this algorithm as ControlDependency( ). It takes Π-ADL as input and gives the output as *ControlDependencyList.* Control dependency is the relationship between calling and called services.

We have taken two lists *ControlDependecnyList* and *ServiceList* which are empty list (ϕ) initially. *ControlDependencyList* is used to store the control dependence elements exist in the system *"OnLineBookStore"*. This algorithm uses two patterns: *ArchitecturePattern* and *CallPattern* which are given as follows:

ArchitecturePattern :

@"\b(architecture|service)\s+\w+\s+(is)\s+(abstraction\((\w+\s*\w+)?\))";

CallPattern:
@"(call|Call)\s+(?<calledname>(\w+))[\(](\w+)[\)](\.)";

Description of *ArchitecturePattern* is given in the explanation of algorithm *MainProgramtoServiceDependency*. CallPatern is used to extract the pattern of a service call . 'Call' is a keyword in Π-ADL which is used to call a service and *calledname* is a variable to store the name of called service. *ADLProgramPartition* stores the block of Π-ADL which is partitioned on the basis of ArchitecturePattern. Two variables countcontrol and count have been taken for counting control dependency and services respectively. Two *foreach* loops have been used which are nested. First foreach loop as foreach (part∈ADLProgramPartition) in which variable part has been used to take elements from *ADLProgramPartion* one by one in order.

Algorithm3:Proposed Algorithm for Control Dependency Extraction

```
1. Algorithm: ControlDependency( )
2. Input: Π-ADL
3. Output:  The output will list out the service and
   relationship with called services
   i.eControlDependencyList
4. // Starting of Algorithm
5. {
6. ControlDependencyList← ϕ;        // This represents
   that list is empty initially
7. ServiceList← ϕ;                          //at ini-
   tial ServiceList is empty
8. //This pattern is used to extract  ADL Program
   block by block
9. ArchitecturePattern←
   @"\b(architecture|service)\s+\w+\s+(is)\s+(abstract
   ion\((\w+\s*\w+)?\))";
10.      // In ADLProramPartion list different blocks
   of  ADL program is listed
11.      ADLProgramPartition← Split Π-ADL  Program on
   the basis of ArchitecturePattern;
12.  ServiceList←GetAllServices( );//Calling of Get-
   AllServices procedure which is defined at end of
   section
```

```
13.      // Logic for Control Dependency
14.      countcontrol←0;//countcontrol has been used
   as subscript of the controlDependencyList
15.      count←0// count has been used as subscript
   for extracting services from Service
16.      foreach( part∈ADLProgramPartition)
17.      {
18.      callingService←ServiceList[count];         //
   each service is assigned one by one in callings-
   ervice
19.      // this is a pattern for extracting called
   services  by calling service and callPattern has
   been used to extract called services from a block
   of ADL Program
20.      callPat-
   tern←@"(call|Call)\s+(?<calledname>(\w+))[\(](\w+)[
   \)](\.)";
21.      totalcalledService← extract called services
   from parts on the basis of callPattern;
22.      if(totalcalledService.Count>0)
23.      {
24.      foreach(s ∈ totalcalledService)
25.      {
26.      calledService←s.calledname;
27.      ControlDependencyList[countcontrol]←
   callingService + " <--   "+ calledService;
28.      countControl ←countControl+1;
29.      }
30.      }
31.      count← count+1;
32.      }
33.      returnControlDependencyList;
34. }//end of algorithm
```

6.4 'DATA DEPENDENCY' EXTRACTION ALGORITHM

This algorithm is used to analyze the data dependency among services in a SOA based system. This algorithm takes Π-ADL as an input and generates list of data dependencies among services on the basis of data dependency patterns defined in 'call' Pattern. As starting of the algorithm there is no element in the *DataDependencyList* thus we initialize it empty (ϕ). This algorithm mainly checks the service call and return. When a service calls to another service, in that case invoking service may send some data to the invoked service and invoked service may return the data to the invoking service.

Algorithm 4: Proposed Algorithm for Extraction of Data Dependency

```
1. Algorithm: ControlDependency( )
2. Input: Π-ADL
3. Output:  The output will list out the service and re-
   lationship        with        called        services
   i.eControlDependencyList
4. // Starting of Algorithm
5. {
6. ControlDependencyList← ϕ;          // This represents
   that list is empty initially
7. ServiceList← ϕ;                              //at ini-
   tial ServiceList is empty
8. //This pattern is used to extract  ADL Program block
   by block
9. ArchitecturePattern←
   @"\b(architecture|service)\s+\w+\s+(is)\s+(abstractio
   n\((\w+\s*\w+)?\))";
10.  // In ADLProramPartion list different blocks of
   ADL program is listed
11.  ADLProgramPartition← Split Π-ADL   Program on the
   basis of ArchitecturePattern;
12.    ServiceList←GetAllServices( );//Calling of Get-
   AllServices procedure which is defined at end of sec-
   tion
13.  // Logic for Control Dependency
14.  countcontrol←0;//countcontrol has been used as sub-
   script of the controlDependencyList
15.  count←0// count has been used as subscript for ex-
   tracting services from Service
16.  foreach( part∈ADLProgramPartition)
17.  {
18.  callingService←ServiceList[count];        //  each
   service is assigned one by one in callingservice
19.  // this is a pattern for extracting called services
   by calling service and callPattern has been used to
   extract called services from a block of ADL Program
20.  callPat-
   tern←@"(call|Call)\s+(?<calledname>(\w+))[\(](\w+)[\)
   ](\.)";
21.  totalcalledService← extract called services from
   parts on the basis of callPattern;
22.  if(totalcalledService.Count>0)
23.  {
24.  foreach(s ∈ totalcalledService)
25.  {
26.  calledService←s.calledname;
27.  ControlDependencyList[countcontrol]←
   callingService + " <--   "+ calledService;
```

```
28.  countControl ←countControl+1;
29.  }
30.  }
31.  count← count+1;
32.  }
33.  returnControlDependencyList;
34.  }//end of algorithm
35.  Output:  The output will list  out the service and
   relationship with the services on the basis of mes-
   sage passing i.eDataDependencyList
36.  // Starting of Algorithm
37.  {
38.  DataDependencyList←ϕ;           // This represents
   that list is empty initially
39.  ServiceList←ϕ;                              //at ini-
   tial ServiceList is empty
40.  //This pattern is used to part ADL Program block by
   block
41.  ArchitecturePattern←
   @"\b(architecture|service)\s+\w+\s+(is)\s+(abstractio
   n\((\w+\s*\w+)?\))";
42.  ADLProgramPartition← Split Π-ADL   Program on the
   basis of ArchitecturePattern ;
43.    //Calling of GetAllServices procedure which is
   given in last of this section
44.    ServiceList←GetAllServices( );
45.  // Logic for Data Dependency
46.  countdata←  0;
47.  count← 0;
48.  foreach( part ∈ ADLProgramPartition)
49.  {
50.  callingService←ServiceList[count];        //  each
   service is assigned one by one in callingservice
51.  // this is a pattern for extracting called services
   by calling service
52.  callPat-
   tern←"(via)\s+(?<return>\w+)\s+(receive)\s+(call|Call
   )\s+(?<calledname>(\w+))[\(](?<pname>\w+)[\)](\.)";
53.  totalcalledService←extract  called  services  from
   parts on the basis of callPattern;
54.  if(totalcalledService.Count>0)

55.  {
56.  foreach(s ∈ totalcalledService)
57.  {
58.  calledService←s.calledname + "@ Parameter= "    +
   s.pname + " @Return Value =" + return ;
59.  DataDependencyList[countdata]←     callingService +
```

```
    "      ←"+ calledService;
60.   countdata←countdata+1;   }

61.   }
62.   Count=count+1;
63.   }
64.   Return DataDependencyList;
65.   } //end of algorithm control dependency
```

part∈ ADLProgramPartition, foreach loop continues until ADLProgramPartition exists the item. 'part' variable belongs to the the ADLProgramPartition. CallingService initially takes the service from the ServiceList, each time as it increases the outer foreach loop. CallPattern mentioned in the algorithm checks the pattern matching cases in the Π-ADL, which is shown and explained below.

callPattern←"(via)\s+(?<return>\w+)\s+(receive)\s+(call|Call)\s+(?<calledname>(\w+))[\(] (?<pname>\w+)[\)](\.)";

'via' keyword comes before any other word when there is the case of service call. After via keyword there has to be at least one or more blank spaces denoted by '\s+' . Next, return variable holds the data value returned by the called service after execution that service as last statement. 'receive' keyword comes after some spaces and followed bysome spaces. After that, call keyword is there and some spaces. And then, '*calledname*' variable stores the called service name of type any word of at least length one and can have any character or numeric value. Finally, *pname* stores parameter name of any alphanumeric type of minimum one length.

We have taken a variable *totalcallService* which extracts the text from *part* on the basis of callPattern. If only number of *totalCallSerive*is greater than zero then inner foreach loop executes. In inner foreach loop, variable 's' is used to extract each and every element from *totalcalledService* one by one order. The called service is assigned by variable s with property callednamem, pname and return defined in the calledPattern as name of called service, parameter and return value respectively. Inner foreach loop continues until and unless there exists an item in *totalcallService*. *DataDependencyList* which stores the data dependency by reverse pointing from calling service to called service as inner foreach loop exists. Finally, after reading all items from *ADLProgramPartiton, DataDependencyList*is returned.

6.5 'SEQUENTIAL DEPENDENCY' EXTRACTION ALGORITHM

This algorithm calls the ControlDependency algorithm for the list of control dependency items. On the basis of control dependency items, we can observe the sequence dependency if service A calls to service B, service B calls to service C and so on. In that situation we have sequence dependency among services because there is a sequence of calls among A, B and C.

The algorithm Sequence Dependency gets ControlDependencyList as input and publishes SequenceDependencyList as output as mentioned in the algorithm. Initially there is no item in SequenceDependencyList thus declared as empty list specified by ϕ. Sequence pattern is as follows is used to extract the call between one services to another service.

sequencepattern← @"(?<parent>\w+)\s+ (←)\s+(?<child>\w+)";

In the above mentioned pattern, variable parent is as calling service and child is called service where child service points to the parent service that means parent service is control dependent on child service thus there is a sequence dependency from parent to child. Variable SequenceService takes the list of ControlDependencyList on the basis of sequencepattern. Outer foreach loop executes until there is an item in SequenceService. At first, sequence variable is declared as empty ,'parent' is assigned as parent property of item and 'child' is assigned as child property of item that belongs to SequenceService. Afterward, sequence is assigned by control relationship where child is pointing to parent. Internal Loop executes when an item1 reads each item of SequenceService.

Algorithm 5: Proposed Algorithm of Sequential Dependency Extraction

```
1. Algorithm: Sequence Dependency
2. Input: ControlDepencyList
3. Output :SequenceDependencyList
4. // Starting of Algorithm
5. {
6. SequenceDependencyList← ϕ;      // initially Se-
   quenceList is empty
7. //Calling of the procedure ControlDependencyi.e all
   control dependency relationships are assigned to
   ControlDependencyList
8. ControlDependencyList←ControlDependency( );
9. // Pattern for sequence on the basis of ControlDe-
   pendencyList
10.      sequencepat-
   tern←@"(?<parent>\w+)\s+(←)\s+(?<child>\w+)";
11.      //Logic for Sequence Dependency
12.      SequenceService←getMatches on the basis of
   sequencepattern from ControlDependencyList
13.      j=0;
14.      foreach(item ∈SequenceService)
```

```
15.       {
16.       sequence←     ϕ;
17.       parent←item.parent;
18.       child←item.child;
19.       sequence←    parent + "← " +child;
20.       foreach( item1∈ SequenceService)
21.       {
22.       if(item1!=item)
23.       {
24.       gchild←item1.child;
25.       gparent←parent;
26.       parent←    item1.parent;
27.       if(child = parent)
28.       {
29.       sequence←  sequence+ "    ←       " +gchild;
30.       }
31.       child←gchild;
32.       }
33.       }
34.       SequenceDependencyList[j]  ←  sequence;
35.       j←  j+1;
36.       }
37.       returnSequenceDependencyList;
38.       } // end of the algorithm
```

If item1 is not equal to item then gchild is assigned by child property of item1. gparent is assigned by parent and parent reassigned by parent property of item1. If child of previous item is equal to the parent of the current item.then there is sequence dependency between previous sequence dependency and the gchild where gchild points to previous. Then, child is assigned by gchild. As inner loop gets out then SequenceDependencyList is assigned by sequecnce. Just before exiting the outer loop, all sequence dependencies are assigned to SequenceDependencyList. Finally, SequenceDependencyList is returned as output to the invoking service.

6.6 ALL SERVICES EXTRACTION ALGORITHM

This algorithm extracts all the services exists from Π- ADL document of 'Online BookStore' system.

Algorithm 6 : Proposed Algorithm for All Services extraction

```
1. Algorithm :GetAllServices( )
2. Input: Π-ADL    Program
3. Output: AllServicesList
```

```
4. //Start of the Algorithm
5. {
6. allservices← ϕ;          // allservices list has
   no element  at initial
7. // Pattern for service extraction
8. // ArchServPattern has been used  to extract the
   service without logic and service name
9. ArchServPattern←
   @"\b(architecture|service)\s*(?<name>\w+)\s*(is)\s*
   (abstraction\((\s*\w+\s+\w*)?\))";
10.      // archServCollection list stores the ser-
   vices  from Π-ADL program
11.      archServCollectiongetMatches of  Π-ADL   on
   the basis of ArchServPattern;
12.      //logic for service extraction
13.      foreach(service ϵ archServCollection)
14.      {
15.      allservices.Add←  service.name;   // service
   name is added to all services
16.      }
17.      return  allservices;
18.      }  //End of algorithm GetAllServices
```

7-DESIGN DESCRIPTION AND INTERFACES OF DEVELOPED TOOL 'DA-SOA'

An attempt has been made to automate the above discussed approaches. A tool named '**DA-SOA(Dependency Analyzer for SOA based System)'** has been developed based on the above proposed algorithms. The proposed tool takes Π-ADL (pi –Architecture Description Language) of a SOA based system as input and gives the different types of dependence relationships, in the SOA based System, as output.

In general, the objective of the developed tool is to facilitate dependency extraction, at architectural level, of a SOA based system. The creation of this tool is guided by the following questions:

- Who will be the probable user of this dependency extraction tool?
- Who will be the beneficiaries of this tool?
- What are the major functionalities provided by this tool?

In response to the first question, a software architect may use this tool 'DA-SOA' to identify dependencies among services. An architect is responsible

for software architecture design. Software tester may use this tool for debugging at software testing phase. They can get information of all service relationships that will help in integration testing and system testing. A software maintainer may use this tool for maintenance purpose at later stage.

For the second question, service provider and service consumer both will be benefitted from this tool. Since service and client applications are evolved independently. So both service and client applications benefit from this tool for understanding the system chaotic situations.. For example, project manager of the proposed system uses this tool and becomes aware of the risk in the system. The main reason for failure of SOA based system is unawareness about dependency existence among services. When developer is aware about dependencies in the SOA based system, then risk can be handled easily during development or implementation of the system. Identification of all relationships among services will help to add/delete/modify any service in the SOA based system. The possible effect of modification of one service over the other services can also be analyzed. This tool can also be used for configuration management phase.

This tool will show the different types of dependency in text form, matrix form, and graphical form. This tool has used Π-ADL documentation of a SOA based system as input and through different logic(algorithm mentioned in previous section) generates different types of dependencies as MainProgram to Service Dependency, Composite Dependency, Control Dependency, Data Dependency, and Sequence Dependency. Thus, this tool supports a number of functionality to improve the quality of the proposed SOA based system.

7.1 DESIGN OF THE PROPOSED TOOL 'DA-SOA'

In this subsection, design description of the developed tool 'DA-SOA' has been explained. UML diagrams have been used to show the design of this tool pictorially. We have used 'activity diagram', 'component diagram' and 'deployment diagram' to show the design of the tool with three different aspects. The overall system design includes different modules to extract different types of dependency from architecture description written in Π-ADL.

7.1.1 Activity Diagram

Activity diagram is an important diagram in UML to describe dynamic aspect of the system. It is basically a flow chart to represent a flow from one activity to another activity. Figure 6 shows the various activities of the Dependency Analyzer (DA-SOA) tool for SOA based system. First step is to extract architecture description from an input file then performed pattern matching activity on the basis of architecture pattern, compose pattern and service pattern activity. After the activity of pattern matching, dependency extraction program activity is done. Activities of different types of dependency extraction are done after the activity general dependency extraction. Finally every dependency is displayed in text, matrix and graph form.

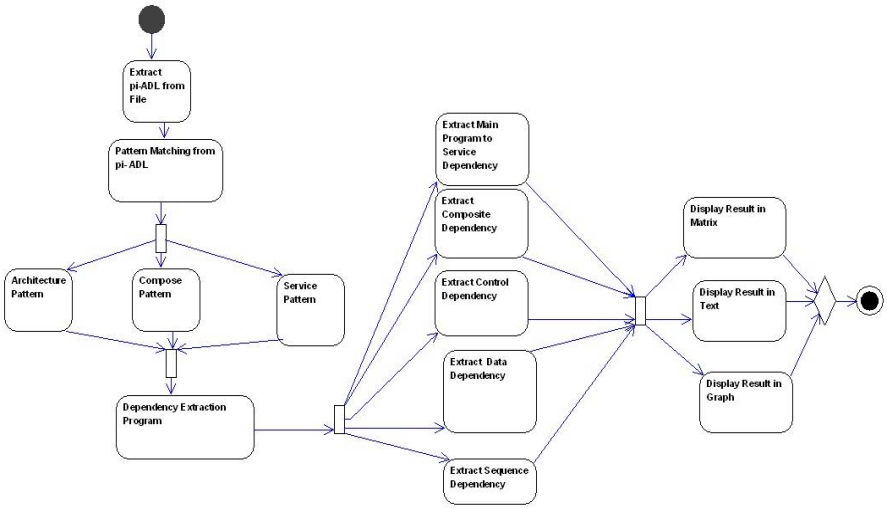

Figure 6. Activity Diagram of 'DA-SOA'

7.1.2 Component Diagram

Component diagrams are used to model physical aspect of a system. This diagram has been used to visualize the organization and relationship among components in a system. The figure 7 depicts component diagram of tool 'DA-SOA'. This diagram shows the dependency among different components. We have taken different modules for tool as in depicted in the following figure.

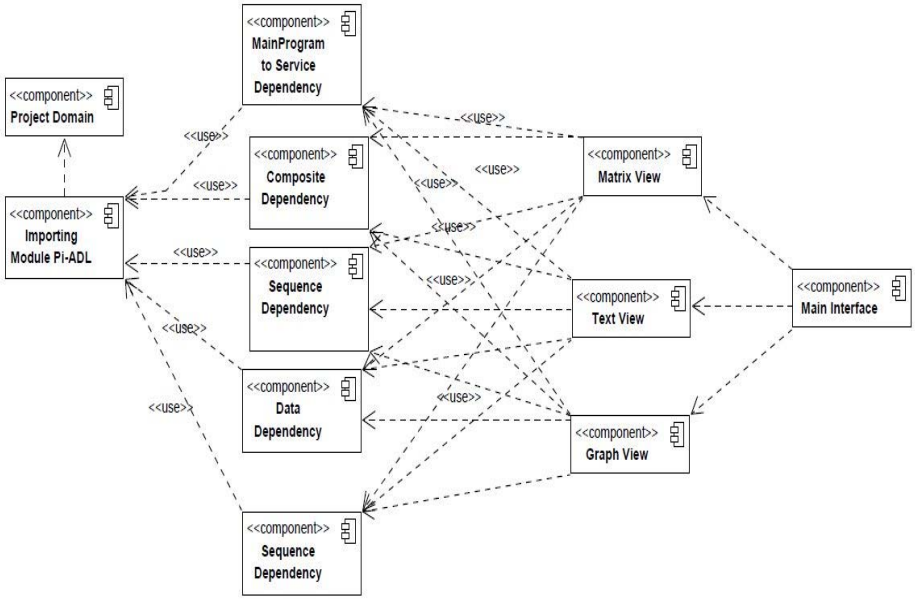

Figure 7. Deployment Diagram of 'DA- SOA'

7.1.3 Deployment Diagram

Deployment diagrams are used to describe the static deployment view of a system. This diagram consists of nodes and their relationship. This diagram is similar to component diagram as depicted in the following figure. The diagram in figure 8 shows the deployment component diagram for development of **DA-SOAtool**.

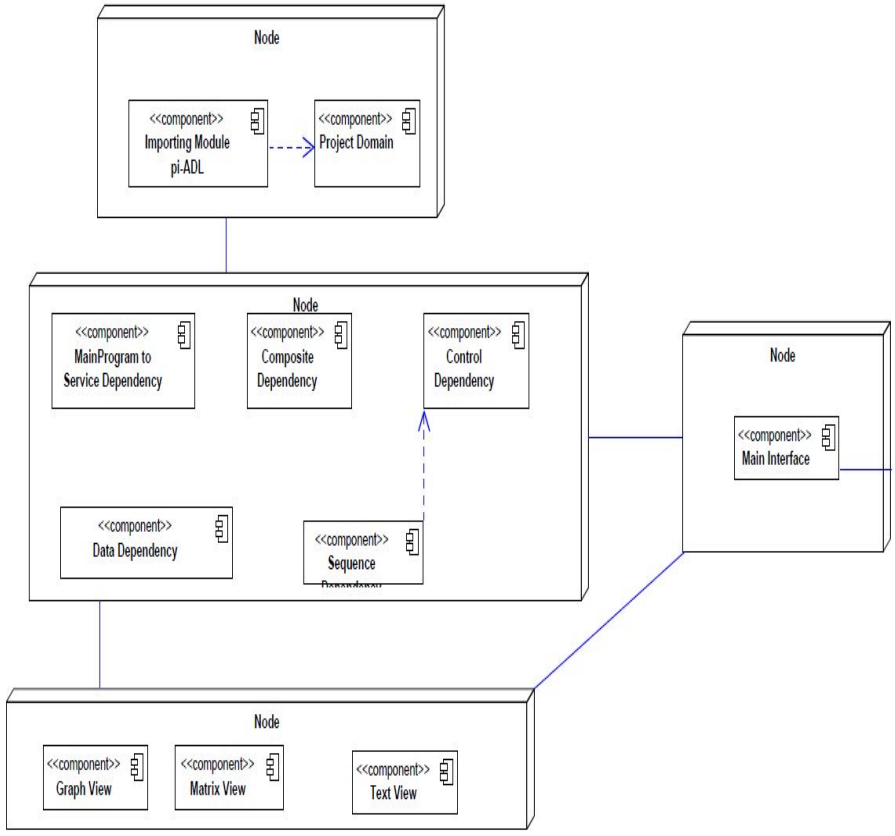

Figure 8. Deployment Diagram of 'DA-SOA'

7.2 DESCRIPTION OF THE TOOL DA-SOA

The tool 'DA-SOA' has been developed in C# using .NET framework. The tool 'DA-SOA' reads the file of architecture description document written in Π-ADL of a SOA based system (which is given in appendices). The tool generates different dependencies, at architecture level, in the system. This tool has 'main interface' by which a person can visualize different types of dependencies in text view, matrix view and graph view. Snapshot of main interface of this tool has been shown in the figure 9. By this interface, to view dependency

in any form, we use the menu bar item and get the result in different view. We have given the detail of different dependencies and snapshot generated by this tool in this section.

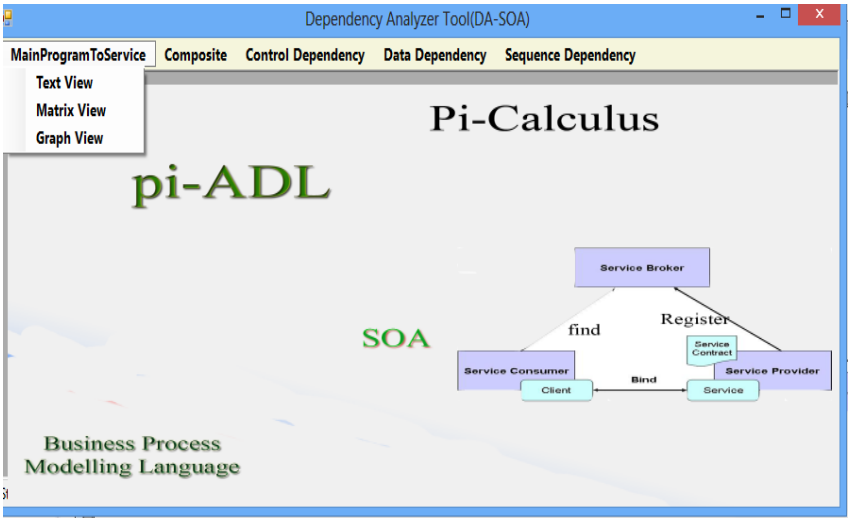

Figure 9. Main Interface of the tool DA-SOA

Generated dependencies by this tool *are Main Program to Service Dependency*, *Composite Dependency*, *Control Dependency*, *Data Dependency* and *Sequence Dependency* in different format. In the graph view, blue rectangle has been taken as atomic service and green rectangle as composite service. Black arrow represents dependency; red arrow shows the parameter passing and green arrow show the value return by the service. Violet row show control passing. We have taken composite dependency as example for the purpose of demonstration of tool.

Screen snapshots of different view of Composite Dependency using 'DA-SOA' are demonstrated below.

Composite Dependency Identification through 'DA-SOA'

Composite service is created from individual services to fulfill business requirements. Here composite service depends on individual service. Some changes or removal of individual service can badly affect to composite service. So analysis of composite dependency is inevitable for avoiding risk of catastrophic situation of SOA based system.

(a) Text View

Show TreeView Button displays composite services and other services in

TreeView Control in right side. In the TreeView control relationship between composite service and individual services have been depicted on different level. In the figure 10 composite service *Search* is composed of *Search-ByBookName*, *SearchByISBN* and *SearchByAuthorName.*

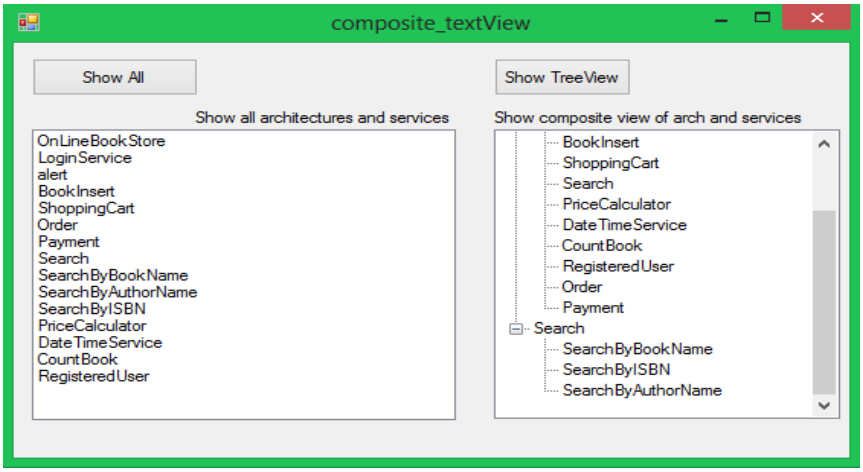

Figure10. Composite Dependency in Text View

(b) Matrix View

m×m matrix with value 0 and 1 where 0 means no dependency and 1 represents dependency from service in row to service in column shown in figure 11, where m denotes the number of services . All the diagonal value is 0 means there is no self-dependency. Row service is the composite form of all the 1's column service. For example, OnLineBookStore is the composite service and it has *LoginService, alert, BookInsert, ShoppingCart* etc. are as membership services. *GridView* Control has been used to show composite dependency of services in matrix view.

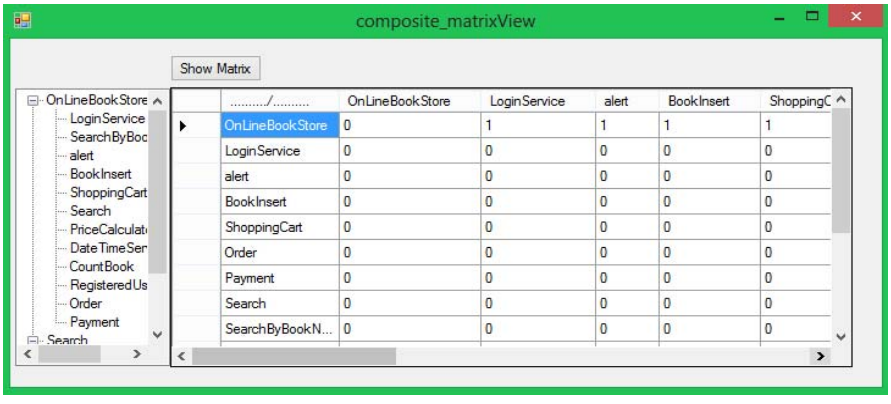

Figure 11. Composite Service Dependency in Matrix View

(c) Graph View

If arrow goes from service A to service B, C, D etc. then service A is composite service consisting of services B, C, D etc. Mathematically A→ {B, C, D...} means A is the composite service consisting of services B, C, D… Figure 12 presents graph view of composite dependency of the system 'OnLineBookStore'. As given below, OnLineBookStore and Search are composite services represented by green rectangle box where other services by blue rectangle box. Service Search points to services SearchByISBN, SearchByBookName and SearchByAuthorName that means Search service is a composite service.

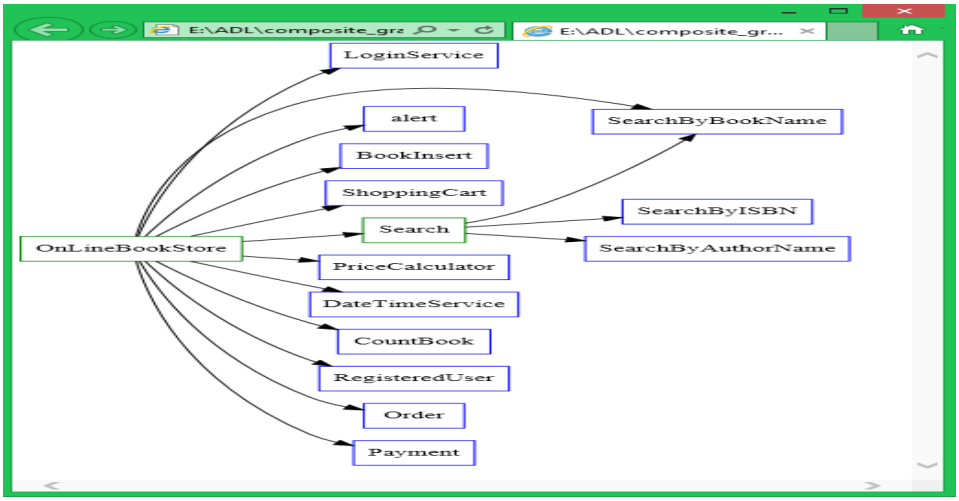

Figure12. Graph View of Composite Dependency

8- CONCLUSION

In the present work, we concentrate on automated dependency identification from an architecture description of a SOA based system. Dependency identification and analysis is relevant research area in SOA based systems. We proposed various algorithms and also describe how we implemented our approach. This tool 'DA-SOA' can efficiently be used for software understandability. Dependency identification in a SOA based system is however a complicated task. Although we have tested this tool for a prototype SOA based system, we are planning to apply it for some real time application and acquire feedback for the further improvement of the tool.

REFERENCES

[1] J. A. Stafford, A. L. WOLF. (2001). Architecture-Level Dependence Analysis for Software Systems, International Journal of Software Engineering and Knowledge Engineering, Volume: 11, Number: 04, August 2001, pp.1-18.

[2] J. Zhao. (1997). Using Dependence Analysis to Support Software Architecture Understanding (1997), New Technologies on Computer Software, 1997, pp. 135-142.

[3] S. Wang. (2010).Dependency Based Impact Analysis Framework for Service-Oriented System Evolution,PhD Thesis, University of Western Ontario, Ont., Canada,2010.

[4] J.A. Stafford, A.L. Wolf and M. Caporuscio. (2003). The Application of Dependence Analysis to Software Architecture Descriptions. Lecture Notes in Computer Science, Vol. 2804 Bernardo, Marco; Inverardi, Paola (Eds.) 2003, pp. 52-62.

[5] M. Rosa and A. O. Sampaio. (2013). SOA Governance through Enterprise Architecture, Web Article. Access at:http://www.oracle.com/technetwork/articles/soa/rosa-sampaio-soa-gov-2080776.html.

[6] D. Garlan, R. Monroe, and D. Wile. (1997). ACME: An Architecture Description Interchange Language. In Proceedings of CASCON'97, Toronto, Ontario, November 1997, pp. 169-183.

[7] D. C. Luckham, J. J. Kenney, L. M. Augustin, J. Vera, D. Bryan, and W. Mann. (1995). Specification and Analysis of System Architecture Using Rapide." IEEE Transactions on Software Engineering, vol. 1, no. 4, pp. 336-355, April 1995.

[8] M. Shaw, R. DeLine, D. V. Klein, T. L. Ross, D. M. Young, and G. Zelesnik. (1995). Abstractions for Software Architecture and Tools to Support Them. IEEE Transactions on Software Engineering, vol. 21, no. 4, pp. 314-335, April 1995.

[9] G. Abowd, R. Allen, and D. Garlan.(1993). Using Style to Understand Descriptions of Software Architecture. In Proceedings of the First ACM SIGSOFT Symposium on the Foundations of Software Engineering, pp. 9-20, Los Angeles, CA, December 1993.

[10] F. Oquendo. (2004). Π-ADL- An Architecture Description Language based on the Higher Order Typed π-Calculus for Specifying Dynamic and Mobile Software Architectures. ACM Software Engineering Notes, Vol.28, No. 8, USA, May 2004.

[11] R. Tolksdorf (2003) A Dependency Markup language for web services. In: Web, Web-Services, and Database Systems, Springer Berlin Heidelberg. pp. 129-140.

[12] C. Ensel, A. Keller (2001) Managing application service dependencies

with XML and the resource description framework. In: Proceedings of IEEE/IFIP International Symposium on Integrated Network Management. doi: 10.1109/INM.2001.918072 pp. 661-674.

[13] S. Basu, F. Casati, F. Daniel (2008). Toward web service dependency discovery for SOA management.Proceedings of IEEE International Conference on Services Computing. , doi: 10.1109/SCC.2008.45, Vol 2 pp.422-429.

[14] T. Espinha, A. Zaidman and H. Gross. (2012). Understanding the Runtime Topology of SOA Systems, 19th Working Conference on Reverse Engineering (WCRE), Kingston, ON, Canada, October 15-18, 2012.pp. 187-196.

[15] SOA Dependency Analyzer 1.0. Available at http://code.google.com/p/bpel-esb-dependency-analyzer/. Accessed on 24th May 2013.

[16] Omer AM, Schill A (2009) Dependency Based Automatic Service Composition Using Directed Graph. In: Proceedings of Fifth International Conference on Next Generation Web Services Practices, Prague, pp 76-81.

[17] E. D. Trigos. (2009). service dependency analysis based on process models and service level agreements. Master thesis, Dresden University of Technology.

[18] J. Brondum, L. Zhu. (2012). Visualising architectural dependencies. 2012 Third International Workshop on Managing Technical Debt (MTD), 5-5 June 2012, Zurich, pp. 7-14.

[19] J. Li, R. Jeffery, K.H. Fung, L. Zhu, Q. Wang, H. Zhang, X. Xu, (2012). A business processdriven approach for requirements dependency analysis, in: Proceedings of 10th International Conference on Business Process Management (BPM'12), LNCS, vol. 7481, Springer, Tallinn, Estonia, 2012, pp. 200–215.

[20] M. Owen, J. Andraj, (2003). BPMN and business process management, http://www.bpmn.org/Documents/6AD5D16960.BPMNandBPM.pdf.

[21] D. Garlan, R. Allen, and J. Ockerbloom. (1994). Exploiting Style in Architectural Design Environments. In Proceedings of SIGSOFT'94: Foundations of Software Engineering, pages 175– 188, New Orleans, Louisiana, USA, December 1994.

[22] A. Terry, R. London, G. Papanagopoulos, and M. Devito. (1995). The ARDEC/Teknowledge Architecture Description Language (ArTek), Version 4.0. Technical Report, Teknowledge Federal Systems, Inc. and U.S. Army Armament Research, Development, andEngineering Center, July 1995.

[23] J. Magee, N. Dulay, S. Eisenbach, and J. Kramer. (1995). Specifying Distributed Software Architectures. In Proceedings of the Fifth Europe-

an Software Engineering Conference (ESEC'95), Barcelona, September 1995.

[24] P. Binns, M. Engelhart, M. Jackson, and S. Vestal. (1996). Domain-Specific Software Architectures for Guidance, Navigation, and Control. International Journal of Software Engineering and Knowledge Engineering, vol. 6, no. 2, 1996.

[25] D. C. Luckham, J. J. Kenney, L. M. Augustin, J. Vera, D. Bryan, and W. Mann. (1995). Specification and Analysis of System Architecture Using Rapide. IEEE Transactions on Software Engineering, vol. 1, no. 4, pages 336-355, April 1995.

[26] M. Greenwood et al. (2003). Process Support for Evolving Ac-tive Architectures, Proceedings of the 9th European Workshop on Software Process Technology, LNCS 2786, Springer Verlag, Hel-sinki, September 2003.

[27] F. Oquendo. (2008). pi-ADL for WS-Composition: A Service-Oriented Architecture Description Language for the Formal Development of Dynamic Web Service Compositions. In: SBCARS, pp. 52–66 (2008).

[28] M. L.Sanz, Z. Qayyum, C. E. Cuesta, E. Marcos. (2008). Representing Service Oriented Architecture Models Using Π-ADL, Proceedings of the 2nd European conference on Software Architecture,Springer-Verlag Berlin, Heidelberg,pp. 273-280.