INTERNATIONAL JOURNAL OF COMPUTATIONAL MATHEMATICAL IDEAS
ISSN:0974-8652 / www.ijcmi.webs.com / VOL 1-NO3-PP 92-99 (2009)

GENERATION OF PATTERN GRAPH BASED ON ARCHITECTURAL QUERY LANGUAGE

Shaheda Akthar[#], Sk.MD.Rafi[$]

[#]Assoc.Professor, Sri Mittapalli College of engineering, Affiliated to JNTU, Kakinada.
shaheda.akthar@yahoo.com
[$]Asst.Professor, Sri Mittapalli Institute of Technology for women, Affiliated to JNTU, Kakinada,
rafi_527@rediffmail.com

Abstract

In the present Chapter, a query graph is first produced by making use of Architectural Query language. A pattern graph is produced based on that query graph by using its Syntaxes and Semantics.

The architectural query language including
i) Construction of system architecture by taking into account their subsystems and their modules.
ii) For the system properties of subsystems and modules are defined.
iii) Taking into account the constraints of the subsystems and modules

Keywords: *AQL, Component, connector, Domain model, Place holders, Query Graph, Query generation.*

Introduction

1. Architecture query language (AQL)

A query containing a number of abstract components and connectors where each component imports a place holder and exports a matched place holder if any is known as an AQL [1].

Abstract Component: A collection of place holder here a place holder is one which retrieves the searching mechanism is called Abstract component.

Abstract Connector: A connection among the abstract components is called an Abstract connector.

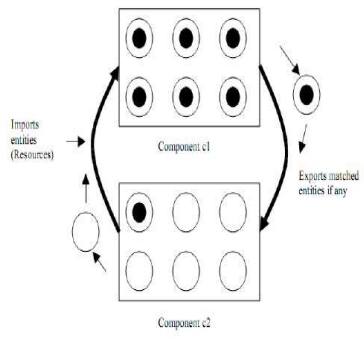


Figure 1: The notations of abstract component and abstract connector in AQL, where the placeholder of abstract component C1 has been matched

The Constraint applied in it should satisfy the number of resources [3] that can be imported and the output should be exported as matched placeholder.

1.1. Notations used in AQL

The notations used in AQL are as follows:

i) **:** means is defined
ii) < > delimits the non terminals;
iii) | means or
iv) << >> denotes optional syntax and
v) [] denotes a set of Instances

1.2. Domain model of AQL:

The attributes involved in AQL query describes the domain model [4] of AQL.

AQL Query Attributes:

Every AQL query is d by a filename (identifier) and it contains many subsystems.

The attributes under AQL are

Name --→ filename

Subsystems → S_1, S_2, ……..,Sn.

Every subsystem is given a unique name, after dividing the system into subsystems and defining the entities that specify the source-region for subsystem.

--The attributes of subsystem are Name and Start node (main seed)

Name → S1/M1

Identify a group of matched placeholders (compared with entities) after giving each sub system a unique name (id)

The following are the attributes of placeholders.

MinimumMatches, MaxMatches, Entities, Imports, Exports

MinimumMatches (<=5): Minimum number of matched placeholders

MaximumMatches(>=10): Maximum number of matched placeholders

Entities: It describes the entities that were matched with placeholders.

Imports: It describes the entities that are to be imported from other components

Exports: It describes the entities(which are matched) that are to be exported to other components.

1.3. STRUCTURE OF AN AQL QUERY:

The structure of an AQL query describes each subsystem attributes and some of placeholder attributes.

Begin

Set of directives;

Begin component S_{s1}

.

.

.

End component S_{s1}

Begin component S_{s2}

.

.

.

End component S_{s2}

Begin component S_{sn}

.

.

.

End component S_{sn}

End.

Step1: Set of directives are used to control the recovery process in the syntax of AQL.

1) One can mention definitely whether the analysis is to be made at file level or at function level

2) One can mention definitely which entities are to be imported and which entities are to be exported.

Step2: The begin-component in an AQL query defines these five keywords.

1. Start node (or) Main seed
2. Imports
3. Exports
4. Exists(hold)
5. Allocate

Begin component: Ss

Main-seed: file 1

file 2

.

.

file n

imports: import others:

Resources from Ss2 (C2)

Resources from Ss3 (C3)

.

.

Resources from Ssn (Cn)

Exports: export others:

Resources from Ss2 (C2)

Resources from Ss3 (C3)

.

.

Resources from Ssn (Cn)

Holds: file Sf1,Sf2,…….Sfn

file 1

file 2

.

.

.

file n

Allocates: Yes →filename→Sd

End component.

A Begin component (subsystem) specifies the number of files, its links to other subsystems from where it can import and to where it can export its simple entities.

1.3.1. Start nodes:

It describes one (or) more files, where each file corresponds to a domain. Select the files that has to be contained in the begin component for the searching process.

Begin component
Start nodes: file 1
file 2
.
.
file n
.
.
.
.
End component.

1.3.2. Imports

An important fragment imports the resources (entities- functions, variables etc). The files of the begin component (or) subsystem uses these files.

Limitations while importing

- Only entities that imported from each subsystem should be in the range (min, max).

Eg., (5, 10)S5 it denotes that a subsystem can import a minimum of 5 resources and a maximum of 10 resources from a subsystem S5.

- The entity should not be imported again the sub system where we previously imported (or) from another sub system if an entity is imported from one sub system.
- An incomplete part imported defines the subsystems from where we import entities. Entities can also be imported from the subsystems that were not defined in the imports fragment. There is no restriction on the size of imports which is the only advantage.

Begin component S_s
Start-nodes: file 1
file 2
.
.
.
file n
Imports: import others→ not defined in the component
Resources (min, max) from Ss2
Resources (min, max) from Ss3
.
.
Resources (min, max) from Ssn
.
.
.
End component.

1.3.3 Exports

An exports fragment exports entities after the matching process, the entities that are to be exported should be in the files of the subsystem Ss.

Limitations while exporting

- Entities which are to be exported are defined. The entities to the subsystem that were not defined in the exports can also be exported, Such entities fragments are called EXPORTOTHERS
- There is no restriction on the size of number of exports which is an advantage. The same entity can be exported to another subsystem.
- If an entity is exported to one subsystem, but cannot be imported to the same subsystem again and again.

Begin component:
Start-nodes: file 1
file 2
.
.
.
file n
Imports: import others→ not defined in the component
Resources (min, max) from Ss2
Resources (min, max) from Ss3
.
.
Resources (min, max) from Ssn
Exports: Export others →not defined in the component

Resources (min, max) to Ss2
Resources (min, max) to Ss3
.
.
Resources (min, max) to Ssn
.
.
.

End component.

1.3.4. Holds

Holds fragment describes the files that are selected from the subsystem's domain including the files in the start nodes. The placeholders (in a subsystem) are matched with the start nodes.

Restriction

The files that were selected in restricted to the range(min, max), where min defines the minimum number of selected and max defines the maximum number of files selected.

Min > files selected > max

Begin component:
Start-nodes: file 1
file 2
.
.
.
file n
Imports: import others → not defined in the component
Resources (min, max) from Ss2
Resources (min, max) from Ss3
.
.
Resources (min, max) from Ssn
Exports: Export others → not defined in the component
Resources (min, max) to Ss2
Resources (min, max) to Ss3
.
.
Resources (min, max) to Ssn
Holds: files selected (min, max)
file 1
file 2
.
.
file n
.

End component.

1.3.5. Allocates

The files that are allocated to a destination subsystem are defined as allocates fragment. The file allocation is performed. In order to import the quality of the recovered subsystem, the file allocation is performed, but the constraint on the size of the subsystems will not be satisfied.

Sending a file to destination subsystem and not sending a file to destination subsystem is involved in a allocation process.

Send a file to a destination subsystem if the allocation process is on (yes) otherwise (no) do not send a file to a destination subsystem.

If the allocation process is on(yes) then send a file to a destination subsystem otherwise (no) do not send a file to a destination subsystem.

Begin component:
Start-nodes: file 1
file 2
.
.
.
file n
Imports: import others→ not defined in the component
Resources (min, max) from Ss2
Resources (min, max) from Ss3
.
.
Resources (min, max) from Ssn
Exports: Export others → not defined in the component
Resources (min, max) to Ss2
Resources (min, max) to Ss3
.
.

Resources (min, max) to Ssn
Holds: files selected (min, max)
file 1
file 2
.
.
file n
Allocates: yes → fileSs → file Sd
No
End component.

1.4. DETAILED STRUCTURE OF ARCHITECTURE QUERY LANGUAGE [2]

Begin component:
Start-nodes: File 1
File 2
.
.
.
File n
Imports: Importing
Functions from other components
Data types from other components
Variables from other components
Exports: Exporting
Functions to other components
Data types to other components
Variables to other components
Holds:Functions to other components
Data types to other components
Variables to other components
Allocates: yes → file Ss → file Sd
End component

1.5. Query Generation

By applying any one of the following methods, We can generate an initial Pattern.

(i)Comparing the source code of the system with its architecture.

(ii)Applying a clustering technique

iii) Using the available system architecture document

The objective of these methods is to extract small groups of system entities that represent the core functionality of the system modules. These groups are used to generate an initial Query graph of an AQL query

Now, we got an idea of the AQL specification. Now, the textual specification of an AQL query should be transformed into a graph representation. The group of graphs that are generated from the query-graph during the iterative matching process are defined. The generated graphs are related by recursive graph algebraic equations Now the query is generated. Generate a pattern graph based on that Query. Therefore the textual specification of an AQL query should be transformed into a graph representation. To generate a graph pattern we have to get a basic idea of source graph and a source region**.**

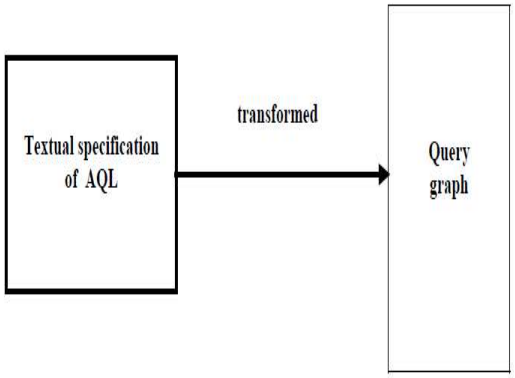


Fig2 Transforming textual specification of AQL to a query graph

1.6. SOURCE GRAPH

- During analysis stage, the source graph is used to model the software system
- The source graph is an attributed relational graph which is used to define all the graphs
- As we know that a graph is a collection of nodes and edges, here nodes represent files, functions, data types and variables and edges represents the relationships. Hence the source graph can be defined as

$G_s=(N_s,E_s)$

1.6.1. SOURCE REGION

Simply the source region is defined as the sub graph of source graph that corresponds to the main seed of G_s where the source region is denoted as G_{sreg}. There is an association property between the source region nodes and the remaining nodes of the source graph (that means the nodes which were not belonged to the source region in the source graph).

E.g.: If a source graph contains the nodes {n1, n2, n3, n4} and if {n1, n3} are the nodes in the source region

then there is an association property between the main seed of the source region and {n2, n4}

The user selects a main seed during the matching phase. Here the matching process at nth phase is defined as

$SR(G_{nphase} = SR(N_{nphase}, E_{nphase}))$

1.7. Graphs based on AQL pattern

1.7.1. Query Graph

A Query graph is an attributed relational graph with query nodes and query edges and is represented as

$G_q = (N_q, E_q)$ where N_q :{qn1,qn2…….qni) is the set of query nodes and E_q: {qe1, qe2, qe3,…….qej) denotes the set of edges in a Query graph

Query node

Partitions can be done at 2 levels file level or at functional level. At the file level the system is divided into number of subsystems and at the functional level it is divided into modules. A query node is defined as an instance of a class module or an instance of a class subsystem. This implies class module at functional level and class subsystem at file level.

Attributes of a query node

- Main seeds
- Name of the AQL query module
- Nodes of the source region

Query edge

A query edge E_q is defined as a link between two nodes qni and qn(i+1).

One node can be used again and again for linking to get an edge.The Attributes of a query edge are group-id for the group of edges and a cardinality range(min, max)

1.7.2. Pattern region

The pattern region is generated by expanding the node of the Query graph G_q. Each Query node is expanded into a pattern region and each query edge is expanded into edge-bundles. The pattern regions can be generated by defining the placeholders where a place holder denotes maximum number of entities specified for a component in an AQL query. Placeholders can be obtained by expanding a query node by keeping some restrictions on it. The pattern region can be represented as

$G_{preg} = (N_{preg}, E_{preg})$

Simply, the nodes and edges between those nodes of the place holder and the nodes and edges of the selected source region are to be matched at matching phase.

Fig 3 Generation of Query graph

In the above figure the query graph is generated with 6 composite nodes from the AQL query text. The edges qe1, qe2 …qe11 denotes the importing and exporting of resources. In the next step the query graph is used to generate a pattern region by expanding the node of a Query graph. By expanding the node qn1, pattern region Gpreg1is generated. By expanding the node qn2, pattern region Gpreg2 is defined and so on…

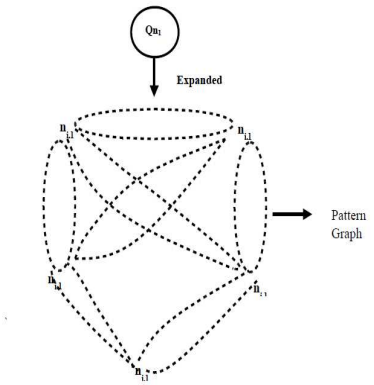
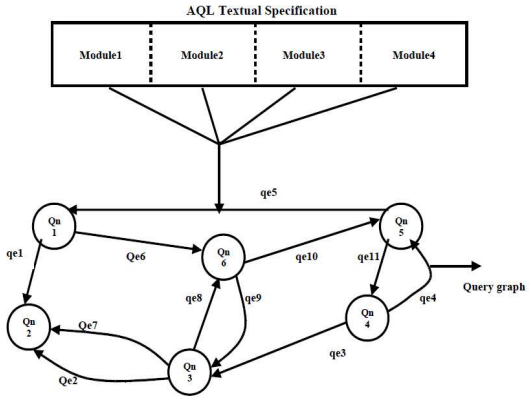


Fig 4 Pattern region generation

The pattern region creates a fully connected graph for any given subset of nodes in a pattern region by calling the functions each other because the functions use the same group of data types and variables.



1.7

.3. MATCHED REGION

The matching process will be performed in n phases until an approximate matching region is found. The matched regions can be found by matching a pattern region with source region Gsreg at a phase such that the constraints while

importing and while exporting should be satisfied

The matched region should be found before the recovering the module of the sub system. The Matched region is denoted as

Gmreg = (Nmreg, Emreg)

EDGE BUNDLES:

While expanding the AQL node from a Query graph a set of

edges may be produced at matching phase i. Simply, an edge bundle connects every node in a

matched region Gmreg to nodes in the pattern region Gpreg

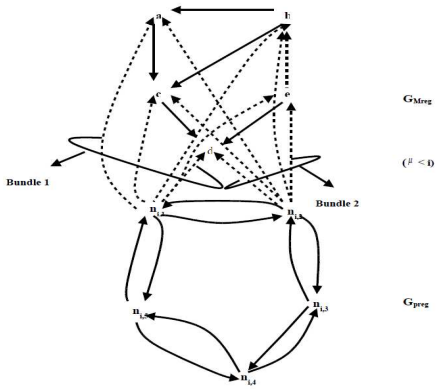


Fig 5 Exported edge bundles

While importing or exporting, it should follow the constraints defined on it. The constraints may functional, data type and variable. The constraint here generally implies the restriction on importing and exporting resources. The Matched region should be found before recovering the module. Edge bundles are generated while importing and exporting resources.

1.7.4. SUMMATION OF GRAPHS

Graph Summation is defined as "If two graphs G1 and G2 are connected the G1+G2 is a disconnected graph with two components G1 and G2.

In this thesis, two operators are defined to compose different graphs. The two operators are

1. +
2. ◯

First operator is used to compose two graphs and the second operator is used to compose a graph and set of edges. Therefore a graph is defined as

$G = (G1 \quad R^{G1->G2} \bigcirc G2$

Or

$G = G1 + (R^{G1->G2} \quad G2) \quad \bigcirc$

1.7.5. MATCHED GRAPH

When the matching process is applied on pattern graph and the input graph

then the matched graph is generated which is a sub graph of a source graph. The matched graph can be represented as

Gm(i) = matchedgraph(Gp, Gi)

Where Gm(i) indicates matched graph obtained during matching process at phase i.

Gm(i) can also be generated with the following equation

$Gm(i) = Gm(i-1)+(Rm<->mr_i \quad Gmr(i))$

This expression can be expanded as follows

$Gm(i) = (Gmr(1)\circ Rm<->mr_1(1))+ (Gmr(2)\circ Rm<->mr_2(2))+ (Gmr(3)\circ Rm<->mr_3(3)+\ldots\ldots\ldots\ldots\ldots\ldots+ (Gmr(i)\circ Rm<->mr_i(i)$

Gm(i)=summation( )

Here m < i as we previously described that the matched region should be found before recovering the module. $Rm<->mr_m$ denotes the imported/exported links from/to every recovered subsystem/module.By using this notation an user can access a module and its import/export

links. The import/export links be with in the matched graph at every phase in the matching process, considering the first match to last match by following indexing method.

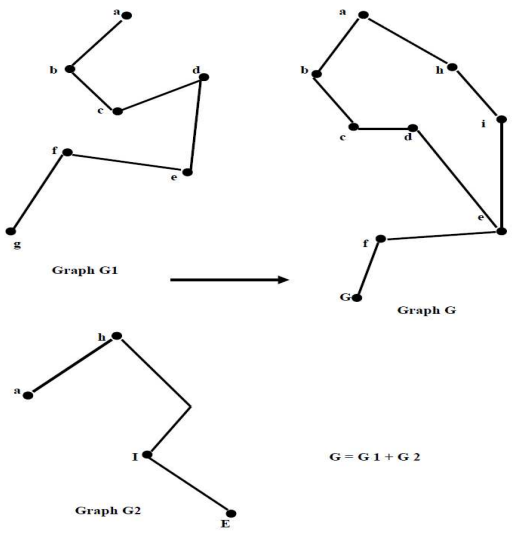


Fig.6 Summation of Graphs

1.7.6. Pattern graph

The pattern graph is generated by expanding the query-graph at different matching phases. At each phase of matching only those Query edges that exist between the current node and the previously matched nodes are expanded. The pattern graph is represented using the graph summation as

$$Gp(i) = Gm(i-1) + (Rm<->pr_i(i) \ 0 \ Gpreg(i))$$

This expression represents at phase i the pattern graph is the sum of the matched graph at previous phase, pattern region at current phase and the imported/exported edge bundles.

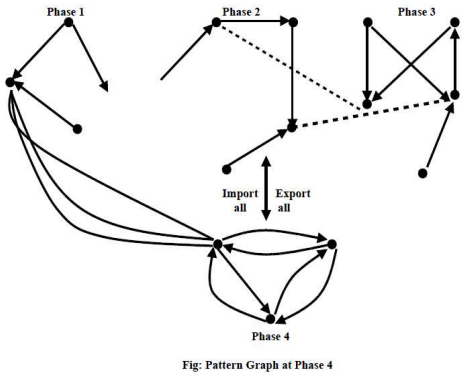


Fig 7 Pattern graph at Phase 4

1.7.7. Input graph

The input graph is a sub graph of the source graph at a particular phase i . The input graph is represented as

$$Gi(i)=Gm(i-1)+(Rm<->sr_i(i) \ 0 \ Gsreg(i))$$

Here Gm(i-1) is the matched graph at previous phase, Gsreg is the selected source region for the current phase i and $Rm<->sr_i$ are edges from the source graph between two sub graphs Gm(i-1) and Gsreg(i). The input graph and pattern graph are supplied to the matching process at phase i.

Acknowledgement
We would like to express our thanks to referees for valuable comments that improved the paper.

References

[1] David C. Luckham and James Vera. An event-based architecture definition language. *IEEE Transactions on Software Engineering,* 21(9):717–734, September 1995.
[2] David Garlan, Robert Monroe, and David Wile. Acme: An architecture description interchange language. In Proceedings of CASCON'97, pages 169–183, 1997
[3] John. A. Zachman. A framework for information systems architecture. *IBM Systems Journal,* 26(3):276–292, 1987.
[4] Philippe B. Kruchten. The 4+1 view model of architecture. *IEEE Software,* 12(6):42–50, 1995
[5] R. Fiutem, E. Merlo, G. Antoniol, and P. Tonella. Understanding the architecture of software systems. In *Proceedings of the 4th Workshop on Program Comprehension,* pages 187–196, 1996