Global Journal of Computer Science and Technology
Volume 11 Issue 12 Version 1.0 July 2011
Type: Double Blind Peer Reviewed International Research Journal
Publisher: Global Journals Inc. (USA)
Online ISSN: 0975-4172 & Print ISSN: 0975-4350

Identifying and Separating Interaction Concerns from Distributed Feature Components

By Vishal Verma,Ashok Kumar
Kurukshetra University P.G Regional Centre, Jind

Abstracts - Implementation of distributed applications over the internet needs the interaction among homogenous/ heterogeneous subcomponents to a great extent. This interaction among heterogeneous components can be implemented by considering the semantic issues of its related compositions. The coordination and cooperation between services at the two ends of application make the problem of interaction more significant. The term interaction can formerly be described in terms of features and services of the application or of the subcomponents & can be called the problem of "feature interaction". This paper proposes a less complex method that uses two concerns termed as signature and transformation. The signature describes the specification aspect of a feature i.e name, arguments etc. On the other hand the transformation describe the working aspect of the feature i.e the fundamental code which actually implements interactions and finally make the two features to work together.

Keywords : *Aspect Oriented Programming, feature interaction (FI), FI resolution, feature based development, signature, transformation.*

GJCST Classification : *D.1.5, I.4.7*

Strictly as per the compliance and regulations of:

Identifying and Separating Interaction Concerns from Distributed Feature Components

Vishal Verma[α], Ashok Kumar[Ω]

Abstract - Implementation of distributed applications over the internet needs the interaction among homogenous/ heterogeneous subcomponents to a great extent. This interaction among heterogeneous components can be implemented by considering the semantic issues of its related compositions. The coordination and cooperation between services at the two ends of application make the problem of interaction more significant. The term interaction can formerly be described in terms of features and services of the application or of the subcomponents & can be called the problem of "feature interaction". This paper proposes a less complex method that uses two concerns termed as signature and transformation. The signature describes the specification aspect of a feature i.e name, arguments etc. On the other hand the transformation describe the working aspect of the feature i.e the fundamental code which actually implements interactions and finally make the two features to work together.

Keywords : Aspect Oriented Programming, feature interaction (FI), FI resolution, feature based development, signature, transformation.

I. Introduction

In the last few years the unavoidable interaction among the homogenous and heterogeneous applications has been increased to a great extent. The interaction among heterogeneous applications leads to the co-execution or co-operation of loosely coupled modules/queries of the software. Here the loosely coupled means the components of software which are designed and implemented independently from each other, have no or very less number of shared elements among them. This type of software components may be developed at same time but on different domains or may be developed by different providers/teams or may be developed by the same provider/team but at different times. Sometimes during the development of such components to maintain the quality of service it become necessary to bypass the semantic reliability among them. Adaptive capability must be provided to facilitate the smooth resolution of conflicts which ultimately leads to the co-ordination and co-operation between different feature components. Maintaining the co-operation and co-ordination in the distributed system is very cumbersome task. The feature interaction problems faced by the telecommunication industry are identified in [13]. The shifting of software solutions from stand alone computers to distributedsystems and taking steps towards the cloud computing makes this problem more significant and ubiquitous.

A desired capability or functionality of a particular query of component may be termed as feature. Within a telecommunication system a feature is expressed as "unit of functionality existing in a system and usually perceived as having a self contained functional role" [3]. It is very common tradition in telecommunication system to organize the development of projects, peoples and even marketing by features [14]. Same process is also followed apparently by the Microsoft for developing their software products [10]. Feature interaction problem involves an undesired interaction in which "the behavior of one feature is affected by the behavior of another feature or another instance of same feature" [7].

Though the FI problem is firstly identified in telecommunication industry, yet it is not limited to the domains of telephony industry. Another means of communication like e-mail, pager, messages etc also face the same problem. Feature interaction related aspects in traditional and telecommunication system are well documented in [6] & [7]. Problems of similar type are also identified in a number of miscellaneous examples like multimedia, mobile and internet services as discussed in [4] & [1]. Service composition problem can also be considered as feature interaction problem [5].

Small size features are used as building blocks of distributed systems. Because of presence of a number of features in the system, the interaction problem becomes inevitably complex. The solution suggested in this paper is based on separation of interaction related issues of the features; this separation is done in terms of signature and transformation. It allows the easy plug and unplugs of interaction resolution modules i.e. the feature interaction concern is raised up to the meta-level. The basic terminology of aspect oriented programming [8] is used to implement/describe this concept of meta-level. The method adds an enhancement to the previous work discussed in [9]. The resolution strategies discussed here are the step forward in the previous identification of concern based requirement engineering [2]. This approach can be best used by taking the concept discussed in the [13] as its plate-form.

Author α : Department of Computer Sc. & Applications, Kurukshetra University P.G Regional Centre, Jind

Author Ω : Department of Computer Sc. & Applications, Kurukshetra University, Kurukshetra , vishal.verma@kuk.ac.in

Importantly this paper takes the issues of resolving interaction problem further by considering two resolutions that themselves interact with each other. Operation precedence is also considered as one of the base to propose the solution of interaction problem. The query based communication system is used as a case study here. Any distributed relational data base system is a typical application of an internet based system, which has many interaction and communication related problems. This system has client and server based structure to store the data and support a number of queries to fulfill the user requests. Important consideration about this system is that it allows a relavar (table) to be stored both at server location (updated periodically) and at client location (updated frequently), to make the easy access of data at client locations. Only highly desired relavars are stored at two locations, to reduce the response time of queries. The feature components taken into consideration are FilterQuery, ForwardQuery and ResendQuery. All of these features are supposed to be implemented at client as well as server location, both for single-copied as well as multi-copied relavars and they help to illustrate interesting interaction properties. Here the same basic approach is used as used in distributed data base i.e. client raise a request for data, which may reside on same machine or on any other machine in distributed system. All the queries raised by one location are supposed to be passed through feature processing components unless it gets executed and result is returned. For simplicity routing issues are ignored.

Filterquery : This feature is used to filter the query based on the user's name and its location from where he/she raise it. To further check the query against the granted permissions in the DBMS, the queries are filtered by using the combination of relavar and user name. Rest all queries are passed to proceed further for execution on other parts of data base.

Forwardquery : This feature component is provisioned to forward a query to a new location based on address of current location to reduce the response time. The reason of forwarding of a query may be heavy load on current location or long distance of server from the client location.

Resendquery : This feature component is used for the timeout queries i.e. if the response of the query is not received within specified time period then it may be resend to the same or to any other target location to get the reply from there.

Based on the above feature components, two interactions among them can be identified as:

Case 1: Considering two locations L1 & L2 of distributed system, both of which keeps the copy of relavar R1. Now suppose a query raised by user U1 from location L1 is forwarded by ForwardQuery to location L2 because of heavy load or any other technical problem but if location L2 has different set of constraints for FilterQuery feature for user U1, then he may not be able to get the reply of his/her query. Here the FilterQuery subverts the ForwardQuery.

Case 2: The ResendQuery feature component become active, and resends the query if user U1 at location L1 does not receive the reply of his query. Here again FilterQuery subverts ResendQuery.

In above two cases it is difficult to resolve the FI conflicts which occur because of the following reasons:

1. Feature under consideration belong to and reside at two different locations. Both locations try to achieve their own goals and follow their own interest.
2. The conflict is acute, therefore difficult to reconcile. In favor of any one side might acutely harm another's side interest.

In above discussed Case 1 location L1 is forwarding the query to another location L2, so that result can be achieved with less response time, but in contrast to location L1, the administrator of location L2 may have different set of constraints applied on query raised by the user U1, to keep the data base secure at his location. This situation causes conflicts among the features and is termed as problem of feature interaction.

In case 2 the user resends query because of time out response from location L1, it causes the same problem of feature interaction among the feature ResendQuery and FilterQuery.

This interaction can also be mapped to many other applications whose working depends upon the execution of query by the user. The resolution of this kind of conflicts is inevitably an important part of system if we want to yield better quality of services. However, most of the existing programming paradigms force the developers to program any query resolution code into the core functionality of a feature (referred as feature transformation). This type of entanglements of different functional roles can quickly complicate a system, making it harder to maintain and evolve. This type of deep seeking into the implementation architecture has led us to propose a two level architecture for complexity control.

One thing that must be pointed out here is that although the suggestion for resolution of each feature interaction is discussed in this paper, we have no intention to strictly validate them, because focus is on the separation techniques, rather than the feature interaction resolution issues themselves. As an aside, it is also believed that there is no definition of resolution in context of feature interaction, reason behind it is that the resolution on the same feature interaction problem may vary from developer to developer. The sentence like "resolution of an interaction" is very subjective and is very hard to implement. Sometimes it just meets the requirements of features of users other than a sound rationalization. In this paper it is assumed that any

resolution step that is able to implement any acute feature interactions constitute a resolution of that interaction. Therefore the simplest resolution is to disable one of the interacting features. However, real world applications might need a more deliberate resolution so as to improve the quality of service.

II. Separation Of Interaction Concerns

For the proposed framework, the work proceeds with an assumption that every feature has a clear specification of its functionality. It is confine that implementation of specification varies, even though it is generally easy to distinguish the pure feature code. This part of the feature is considered as "transformation" i.e the inevitable part for the implementation of specification (feature).

However in feature driven development, features must clearly be able to work with other features. Since transformation is acutely rigid business logic, it is unable to adapt itself to different execution contexts (different interactions among features). Hence the corresponding signatures are required to make the interaction easier and make it flexible enough to adopt with other interacting features. Therefore a feature signature is responsible for gluing features together and taking actions to smooth among incompatibilities.

It is not possible for a developer/designer to foresee the feature that will interact with his/her developed feature, hence signature must be able to adjust with transformation at any stage latter on i.e complementing the situations which causes problem with interaction resolution issues when transformation is designed. To make this kind of implementation possible, it is ideally raised up to the meta-level so as to provide a separation among signature and transformation and facilitate reuse and maintenance/ evolution.

Actually it is signature that is thought to be ideally suited to aspect oriented software development techniques. Fig 1 shows the FilterQuery's transformation part.

```
class filter implements Qrt
{
string qryid;
public filter()
 {
    // set statements for filter box
 }
public void receive(Query qry)
{
 filter_feature(qry);
}
private void filter_feature(Query qry)
{
 string sender=qry.location.user();
   if(! Isfilteredlist(sender))
process(qry)
   else
      discard(qry);
}
public boolean isfilteredlist(string sender)
{
 string      list=sender.substring(sender.indexof("table
name")+1);
return boolean(list);
}
public void process(Query qry)
{
 //execute the query;
}
public void discard(Query qry)
{
  //discard the query;
}
}
```

Fig 1. FilterQuery's transformation, implements only what specification specifies.

The transformation logic takes care of filtering the incoming query against a filter check list. In order to do this, for an incoming query, it will get the sender's address and check it against the filter list, then decide to either process it or discard it depending upon check list's entry. The Qrt interface, which contains two methods, *receive* and *process* must be implemented for the connection of feature boxes. It can be seen that transformation of a feature is simple, cohesive and highly consistent with its original specification. Typically features have two basic parts:

1. Some data (structure) such as list of filter permissions, list of filter users etc.
2. Some method to operate on data and provide necessary feature logic to implement a service feature.

The signature of the feature is expressed by considering the two features together.

a) Filterquery Vs Forwardquery

The signature of the feature can help in an easy way to come out of the problem faced during the interaction among the FilterQuery and ForwardQuery. One might use a form to ask a feature owner to specify options/preferences/policies for dealing with interaction so as to collect the basic data for negotiation, while another way might just design a default resolution policy. A simple resolution based on a default policy is discussed here. As ForwardQuery behave passively for this interaction, the policy requires a decision by the FilterQuery. A reasonable default policy of FilterQuery might be "to apply a check on the received query from user U1 whether this query has an entry in the filter list at location L1 if so then location L2 must follow the same check list for processing of query at its location, but if there is no entry in the filter list at location L1 against user U1 then location L2 is free to use its own

filter list for query raised by user U1." Thus the users as well as locations are free to process the query in their own way, though both locations have their own list to filter the queries. Formally this resolution can be described as:

Every time FilterQuery is about to process a query, it should additionally check if the query is from the ForwardQuery, if so, it must demand and follow the same filter list as obtained from the ForwardQuery's location, otherwise, it is free to process the query as per its own filter list. For this implementation ForwardQuery must add a <forward> tag in the content with original sender's location that allows FilterQuery to check against the filtering list.

Fig. 2 shows the implementation of above resolution in aspect oriented language.

```
aspect ForwardQuery()
{
  flist=filterlist.currentloaction();
    void prc(Query qry, boolean fwd)
     {
       If(fwd)
         {
            flist=getfilterlist.prevlocation();
            process(qry,flist);
         }
       else
           process(qry,flist);
     }
}
```

Fig.2 Resolve FI between FilterQuery and ForwardQuery

b) Filterquery Vs Resendquery

The goal of filtering must be clear at the time of resolution among the features. The features FilterQuery and ResendQuery interact in a number of ways to each other. In some cases the FilterQuery is implemented in context of ResendQuery only to avoid the re-execution of already executed query. If this is the case then problem of interaction can be resolved only by keeping record of acknowledgments to the users/locations. Based on this insight, a resolution can be suggested as follows:

Every time FilterQuery is about to execute a query it checks whether the query is received first time or it is received from the ResendQuery, if it is first arrival then execute it otherwise the FilterQuery will first check the list of acknowledgements for already executed queries, if it found an entry it simply discard the execution. To let the FilterQuery know that the query is a resend query, ResendQuery must add a <resend> tag in the sent query.

Fig. 3 shows the resolution for FilterQuery and ResendQuery

```
aspect stop ReExe()
{
   void filter(Query qry)
    {
      boolean resend;
      resend=qry.location.ack;
      if(resend)
       discard(qry);
      else
         process(qry);
    }
}
```

Fig. 3 Resolve FI between FilterQuery and ResendQuery

III. Problem Faced During Composition

Last section clearly shows that by using the basic terminology of aspect oriented programming for representation of feature interaction resolution is an effective way of feature composition. It is also flexible with respect to further evolution of the system. However, FI problems are complicated issues, and a resolution is unlikely to be independent of other resolutions. This is not unexpected, since resolution themselves can be viewed as features, which, of course are prone to interactions.

Both interactions resolutions discussed above require new behavior of advice around FilterQuery. Basically a FilterQuery feature is used or applying a check on the permissions granted to various users/locations. For every raised query, it either processes the query or discard it. Interestingly, there is an antithesis between two resolutions. The first, i.e the case of FilterQuery and ForwardQuery, says that if the query is a forwarded query then the FilterQuery makes a check by using the filtered list against which it is to be filtered. It always demands the filter list from concerned locations which made an extra burdon on the system. This problem can be rescued by keeping the filter list unique or keeping the filtered list at all locations. While second, i.e the FilterQuery and ResendQuery, says that a query is discarded if their exist acknowledgement in the acknowledgement list for the same query, the situation may be that the acknowledgement and result is sent from the server but is not received at the location because of communication channel problem. The resolution for this circumstance is that we must do something to rescue it from being discarded.

When composing the two resolution features together, the problem is: while one resolution require the processing of query the other resolution wants to discard it. The problem can be reduced by considering the comprehensive view in Fig.4 about the queries that are to be processed after forward of resend query. This typical configuration is shown as follows:

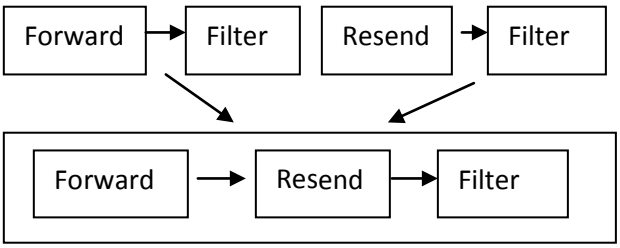

Fig. 4 : Comprehensive view of all features arrangements.

IV. Evolution Of Our Approach

Our evolution can be classified into a number of different properties including cleanness of separation, faithfulness of implementation to specification, re use, adaptability to requirement change etc. Name of all the specified properties shows that they are qualitative than quantitative.

a) Cleanness Of Separation

1. The approach discussed in this paper avoids the tangling of core behavior with resolution code, it allow a feature to work with other features.
2. All the features illustrate an elegant separation when implemented. Important to note is that not every interaction require a separate resolution module thus motivating our search for more general interaction resolution patterns.

b) Faithfulness Of Implementation To Specification:

1. The design of two level architecture keeps the feature's implementation faithful to its specification.
2. This design can be used as a base to generative programming techniques to generate code automatically from specification.

c) Re Use

1. Reuse for very specific interaction resolution modules (Fig. 2 and Fig. 3) is limited. The best opportunity to re use is at the base level (signature) rather than the meta level.
2. The re-factored GUI (if build) modules can be reused in other implementations since all interactions concerns are extracted thus leaving a generic feature component

d) Adaptability To Requirement Change:

1. The separation proposed by our architecture allows the developer to integrate new features into the system, without needing to consider, or worse rewrite, existing feature.
2. The aspect oriented approach for the separate resolution modules allow the developer to implement a feature without considering the interaction with other features, then focus on the interaction issues separately.
3. Removal of a feature from a system to avoid redundant code being left embedded in feature boxes, a situation that leads to unnecessary complexity and low efficiency.

V. Conclusion And Future Work

Heterogeneous service nature of distributed features make the problem of FI in today's applications more sever. It is believed that the separation of interaction concerns is the key to the success of reusability and maintenance of an evolving system. Signature and transformation separation are metaphors for the relationship between a feature's functional logic and its adaption logic. The signature provide a way to transformation so as to allow it to adapt to a feature interaction. Lifting up the transformation code to a meta level is the vital decision for the separation. The emerging area of aspect oriented programming provides a new dimension for the implementation of this concept.

More investigations are required to be carried out to abstract further interaction resolution patterns, and further interaction resolution pattern libraries for different domains. The focus of interaction resolution is the composition problem, namely the semantic conflicts occurring when two interactions resolutions composing together.

References Références Referencias

1. L.Blair, G.Blair., J.Pang. & Efstratiou C. Feature interactions outside a telecom domain. Workshop on *Feature Interactions in Composed Systems,* held at ECOOP2001, Budapest, Hungary, 18-22 June 2001.
2. Kumar Ashok, Verma Vishal. A model for concern based requirement engineering. presented and published in *National Level Seminar*, N C College of Engineering, Panipat, 2008
3. J.Pang, L.Blair. An adaptive run time manager for the dynamic integration and interaction resolution of feature. In proceedings 22nd *International Conference on Distributed Computing Systems Workshops.* pp445-450 IEEE, Los Alamitos, California, 2002.
4. L.Blair. & J.Pang., Feature interactions - life beyond traditional telephony, In [6], pp 83-93, IEEE, Los Alamitos, California 2000.
5. M.Blay-Fornarino, A.Pinna-Dery and M.Riveill. Towards dynamic configuration of distributed applications, In Proceedings 22nd *International Conference on Distributed Computing Systems Workshops,* pp487-492 IEEE, Los Alamitos, California, 2002.
6. M.Calder, E.Magill. editors, Feature interactions in telecommunications and software systems vi. Glasgow, Scotland, IOS Press(Amsterdam), 2000.
7. E.Jane Cameron, Nancy D.Griffeth, Y. Lin, M.Nilson, W.Schnure, and H.Velthuijsen. A feature-interaction benchmark for in and beyond. *IEEE Communications XXXI*(3):64-69, March 1993.

8. G. Kiczales, E. Hilsdale, J. Hugunin, M. Kersten, J. Palm and W. Griswold. Getting started with aspect. Communication of the ACM, Vol. 44, Issue 10, *ACM Press (New York),* pp59-65, October 2001.
9. Kumar Ashok, Verma Vishal. Model for concern data bases. presented and published in *ICACCT* – 2008
10. M.A.Cusumano and R.W.Selby. Microsoft secrets: how the world's most powerful software company createstechnology, shapes markets, and manages people. Simon & Schuster, 1998. ISBN: 0684855313.
11. M Calder, E. Magill, M. Kolberg, and S. Reiff-Marganiec. Feature interaction: a critical review and considered forecast. Accepted for publication, Computer Networks, North-Holland. 2002. Available via http://www.dcs.gla.ac.uk/~muffy/papers.html
12. P.Zave. FAQ Sheet on feature interactions , AT&T, 2001, Available via: http://www.research.att.com-/~pamela/faq.html.