An Adviser for Efficiently Resolve Email Feature Interactions

Fernando Miguel Carvalho
Instituto Superior de Engenharia de Lisboa
Rua Conselheiro Emidio Navarro 1,
1950-062 Lisboa, Portugal
Email: mcarvalho@cc.isel.ipl.pt

Rui Gustavo Crespo
Technical University of Lisbon
Av. Rovisco Pais,
1049-001 Lisboa, Portugal
Email: R.G.Crespo@comp.ist.utl.pt

Abstract

Internet applications, such as Email, VoIP and WWW, have been enhanced with many features. However, the introduction and modification of features may result in undesired behaviors, and this effect is known as feature interaction-FI.

We describe a distributed FI resolution, based on advisers. The adviser follows deontic rules and is implemented with Java technology. In case of failure of one Internet node, message processing is not compromised.

Our proposal was customized and tested to James Email server, and the results satisfy main requirements and do not compromise performance.

1. Introduction

Internet applications have been enhanced with many features, defined as functionality units existing in a system and perceived as having self-contained functional roles [1].

The combination of features may result in undesired behaviours and this problem is known as *feature interaction*, or FI for short [2]. The FI problem, first identified in circuit-switched networks, has been studied in many Internet applications, such as Email [3], VoIP [4] and WWW [5]. The incorporation of Internet applications in Web services increases the number of feature interactions, and such issue is expected to raise concerns in the community.

As a simple example of FI, consider the Email application, where Anne subscribes FilterMessage *and Bob* subscribes ForwardMessage *features, as shown in* ***Figure 1****.*

If Anne sends an Email to Bob, it will be redirected to Charles satisfying the subscribed ForwardMessage *feature. But this behaviour contradicts the* FilterMessage *feature.*

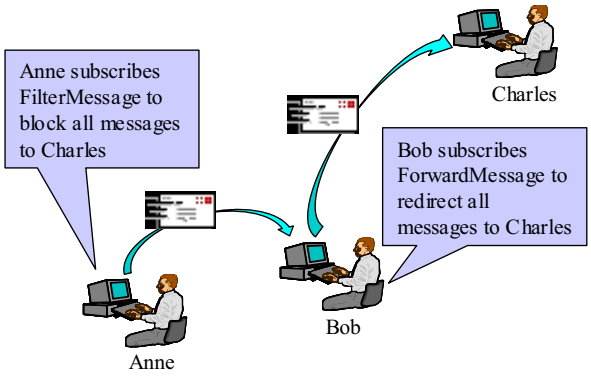

Figure 1. Example of an Email application FI

Three basic problems have been studied in the FI area: *avoidance*, *detection* and *resolution*.

Avoidance means to intervene at protocol or design stages to prevent FIs, before features are executed. The distributed nature of Internet, with multi-provider environments, and the end user capability to program and tailor features makes it impossible to rely on avoidance.

Detection aims at the identification of FIs, with suitable methods [6].

In the resolution, actions are exercised over already detected FIs.

In this paper, we focus on FI resolution based on a scalable Adviser, introduced in section 2, and whose architecture is described in Sections 3 and 4. Section 5 discusses some implementation issues on the Adviser integration in James, a Java Apache Mail Enterprise Server. Section 6 depicts some results of practical experiences on James feature resolution.

Proceedings of the Advanced International Conference on Telecommunications and
International Conference on Internet and Web Applications and Services (AICT/ICIW 2006)
0-7695-2522-9/06 $20.00 © 2006 IEEE

2. Adviser

We propose to attach, to every node in the Internet, one Adviser, which adopts the client-server model as depicted in **Figure 2**.

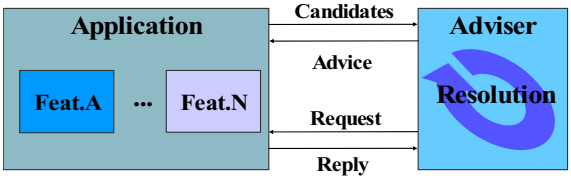

Figure 2. Adviser client-server model

Before executing features, the application sends to the adviser one list of features candidate for execution. The adviser replies with one advise of the feature that application should execute.

For single user FIs, it is sufficient that the local node implements the Adviser. For multiple user FIs, the advisers of involved nodes may have to collaborate in the FIs resolution. The communication capability between Advisers is used, for example, to grant permission for message processing. This is the case presented on **Figure 1**, where Bob Adviser's must have permission from Anne's adviser.

3. Features interaction representation

The number of features may be very large. To make the Adviser as generic as possible, each feature is represented by a set of basic actions. We have selected a small group of basic actions, such as *Deliver*, *Deny*, *Forward* and *Sending* a message. For example, the *AutoResponse* feature, which informs the sender that the message receiver is absent, is represented by the combination of *Deliver(dest)* and *Send(init)* actions. *init* and *dest* references, respectively, message initiator and destination (it may change later, for example as a result of executing Forward feature).

To resolve FIs, we propose the use of constraint formulas. Reasons for our choice includes the similarity of the representation to human knowledge, the easier implementation of the FI resolver, and the successful application of drop actions in the *iptables* IP packet filter [7]. Constraint formulas have the form:

$$Request \wedge Condition \rightarrow Restriction \qquad (3.a)$$

Request is a set of actions, representing features selected by the application as candidates for execution.

Condition identifies the values that the application status must satisfy (for example, the Email application has granted permission for the user to redirect an email), or the adviser must identify (for example, the sender belongs to a specific domain). By default, this part is equal to *true*.

Restriction identifies the single action, or the join of actions, whose execution is forbidden. This part uses the Interdiction operator, *I*.

To resolve the FI case presented in **Figure 1** we use the constraint formula

$$Forward(dest) \wedge \neg permission(init,dest) \rightarrow I\ Forward(dest) \qquad (3.b)$$

4. Architecture

For the Adviser, we identified a set requirements (i) the resolution process should be as abstract as possible (hence, we follow a logic model) (ii) system should be able to dynamically include new features and FIs (iii) communication protocol should be as flexible as possible (iv) communication mechanism should be interoperable, to enable the integration of other systems (hence, we adopted XML oriented) (v) system should be prepared to answer for parallel requests of feature candidates made by different application (hence, it run on a multi thread model) (vi) must be able communicate with another Adviser from a different node, and (vii) architecture should be modular.

Finally, the Adviser must run all this processes without compromise its performance and the application. To solve these requirements we have organized the IR Adviser functions in three main components, *Resolver*, *Interpreter* and *Listener*.

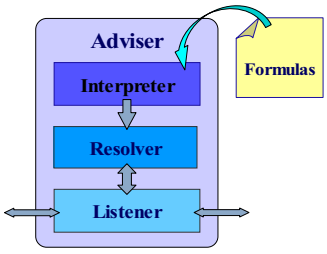

Figure 3. Adviser architecture

Each component has a specific role, which will be explained in the next three subsections. Information about Java class diagrams for the Advisor components is published elsewhere [9].

4.1. Interpreter

Constraint formulas are expressed in a textual language, that adopts first-order predicate grammar [8] plus the Interdiction unary connective *I* (*IA* sub formula means that action *A* cannot he be executed). The EBNF syntax [10] is

formula_list → *formula*$^{+}$
formula → *left "=>" status ";"*
left → *actions | conditions | actions "&&" conditions*
actions → *ACTIONS id_list*
conditions → *CONDITIONS condition*$^{+}$
condition → *"(" ID id_list ")"*
status → *order id_list*
order → *INTERDICTIONS | PERMISSIONS*
id_list → *"(" ID*$^{+}$ *")"*

For example, formula 3.b is expressed in our programming language as:

actions(Forward) && conditions(conf.NoPermission())
=> *interdictions(Forward);* (4.1.a)

Actions are constant identifiers, represented by a `String`. Conditions are represented by class identifiers, and may be initialized with arguments, such as specified by the formula.

actions(Deny) && conditions(conf.FromHost(gov))
=> *interdictions(Deny);* (4.1.b)

The Interpreter's Lexical and syntactic analyzers are implemented with help of JLex and CUP tools [10].

4.2. Resolver

The Resolver is the central component of Adviser architecture. It implements the Adviser core functionality on the identification of an advice (a selected feature) for a list of feature candidates, sent by the application. This process runs around two distinct entities: formulas and candidates, each one represented by instance of appropriate class.

When an application sends to the Adviser a list of feature candidates for execution, Listener processes that message and instantiates a new `candidate`, which will be sent to the Resolver. For every message received by the Resolver, a new thread is launched for the execution of the resolution process, represented by a `Resolution` instance. When this thread finishes, an advice has been determined and it will be returned to the application through Listener.

The resolution process will cover all Formula instances against the `candidate` instance, through an invocation of a `validate` method. The method succeeds when it verifies (i) one correspondence between the message's candidate features and formula actions, and (ii) the formula is satisfied. `validate` method returns a set of interdicted actions, which may be empty.

At the end, Resolver selects one feature among those represented by the survival actions.

4.3. Listener

The implementation of the communication process may use Java RMI [11], due to its simplicity. However, that choice will restrict the integration between applications and the IR Adviser. Therefore, the Java RMI integration with any other technology different than Java would be unrealisable.

To avoid this scenario, we selected an independent application communication solution with sockets [12]. This option implies the specification of an explicit protocol for the exchanged messages. The protocol is specified in XML [13], because of its benefits on (i) textual representation of object's structure and internal state, (ii) serialization and desserialization, into and from Java objects, respectively, (iii) tools are available for XML text analysis and (iv) XML is a standard well accepted in the scientific and industrial community.

JAXB-Java Architecture for XML Binding framework [14] implements the translation process of the XML messages into Java objects. The description of the elements, entities, and relationships of the exchanged messages are specified by a schema file, which is parsed by JAXB binding compiler.

The Listener receives messages from two different entities, applications and IR Advisers of another nodes. To handle connections from both entities, Listener runs two different threads. Depending on the connector being an application or an adviser, the appropriate class is instantiated.

Once the connections are established, the communication between the Listener and the application, or another adviser, will be handled by appropriate class instances.

Listener translates application messages, to `Candidate` instances, which are sent to the Resolver. Finally, Resolver returns to the application its advice. This process is asynchronous and the responses from the Resolver for the requests could be made on a different order.

Messages from other Advisers are posted by their Resolver's, which send a request to be evaluated by the Application of current node. In this case, Listener redirects the request to its Application and returns the response back to the Resolver attached to the remote Adviser.

5. Implementation

James is made of two agents, message transfer and mail processing agents. Message transfer agent moves Email messages between nodes and follows Internet protocols such as SMTP [15], mail message formats and message stored retrieval (POP3[16] and IMAP[17]).

Mail processing agent, depicted in **Figure 4** and defined in the config.xml file, is a chain of *processors*. Processors are *mailet* containers, and each one processes Email messages.

Mailet attributes define conditions for selecting an Email message, and the object class that processes the Email message. If a match occurs, the message is immediately processed by the *mailet*. If not, the message is passed to the following *mailet* in the chain.

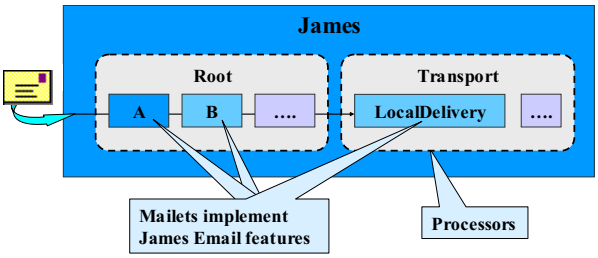

Figure 4. James Architecture

While James executes immediately a matched feature, in our approach the application must wait for the feature returned by Resolver. To integrate the Adviser in James architecture, in the mailets we removed the immediate processing and added two more components:

- *FeaturesManager*, to process all matched features;
- *AppListener*, to support the communication mechanism for the Adviser.

The extended architecture and the configuration files are depicted in **Figure 5**.

6. Results

The Adviser implementation was tested in 1.6 GHz PCs, provided with Windows 2000 and XP operating systems, and connected to a 100 Mbps LAN.

We tested three different FI resolution cases:

- Simple Resolution, based only on actions.
- Conditional Resolution, based on actions and local conditions
- Chain Resolution, based on actions and conditions identified in every node on a remote advisor.

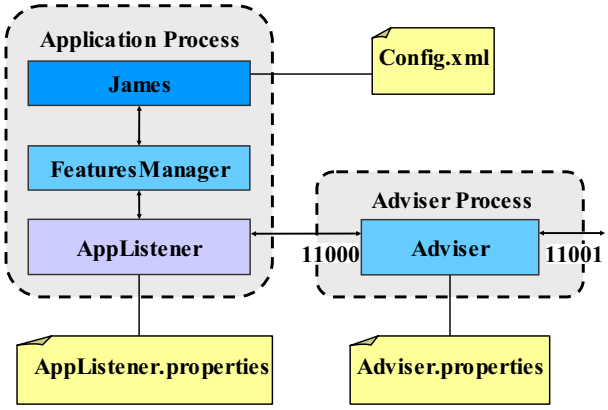

Figure 5. James extended architecture

For each test we have compared the delivery time with, and without, Adviser resolution. Email deliveries suffered varying degrees of degradation, according to the processing cases exercised by Adviser. In the worst case, with request delivery confirmation from the sender, deliver time increased by 55%.

The performance did not decrease in larger extension, because the resolution is linear. The Adviser, resolution and communication architecture are also very efficient.

Surprisingly, in some particular cases, presented in section 6.1, Email delivery was faster with Adviser.

6.1. Simple Resolution

We experimented four different cases with Adviser holding two formulas:

actions(Forward,Deny)=> interdictions(Forward);
actions(Delivery,Deny)=> interdictions(Delivery);

The cases differ from the number and kind of features subscribed on destination. Each case adds one feature to the previous case, and are: (1) no services subscribed, (2) *ForwardMessage* subscribed, (3) 2nd case features plus *FilterMessage* subscribed, and (4) 3rd case features plus *AutoResponder* subscribed.

The results are depicted in **Figure 6**. The light and dark and dark columns represent, respectively, the Email delivery time with and without Adviser.

In cases 1 and 2, the communication between the James server and Adviser takes between 20 and 60 ms. Cases 3 and 4 reveal a decrease in the total processing time, because the communication between James server and Adviser takes less time than the analysis of formulas for the three unnecessary features.

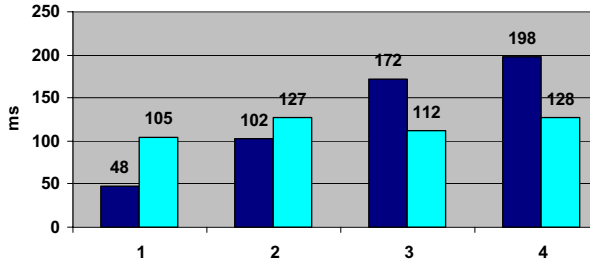

Figure 6. Simple resolution experiments

6.2. Conditional Resolution

We experimented four different cases with Adviser holding four formulas:

actions(Delivery,Send) &&
conditions(conf.FromUser (postmaster))
=> interdictions(Send);
actions(Delivery,Send) &&
conditions(conf.FromHost(yahoogroups.com))
=> interdictions(Send);
actions(Deny) &&
conditions(conf.FromUser(president))
=> interdictions(Deny);
actions(Deny) && conditions(conf.FromHost(gov))
=> interdictions(Deny);

Results, depicted in **Figure 7**, show that Adviser increases the Email delivery time around 50%.

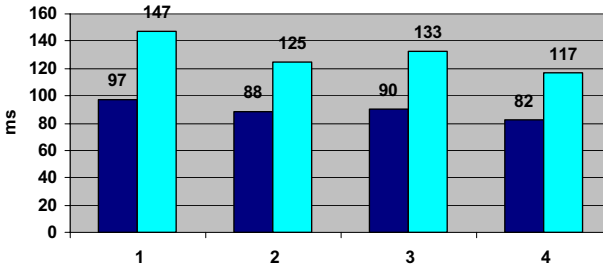

Figure 7. Conditional resolution experiments

6.3. Chain Resolution

We experimented two different cases with Adviser holding two formulas:

actions(Delivery) && conditions(conf.FilterAll())
=> interdictions(Delivery);
actions(Forward) && conditions(conf.AdvLoop())
=> interdictions(Forward);

The condition *FilterAll*, checks if users designated by the destination are listed in the message chain addresses. Second formula detects an infinite loop, caused by the *ForwardMessage* feature redirecting users to each other.

Experimental results are similar to conditional resolution cases.

6.4 Remote Resolution

We experimented the case on formula (4.1.a). We registered an increase of 55 % in the Email delivery time, as depicted in **Figure 8**.

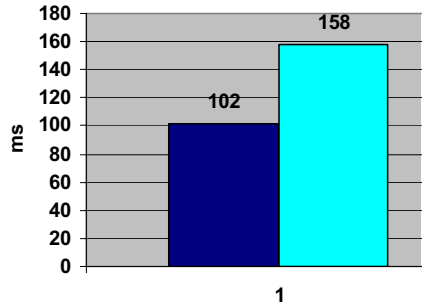

Figure 8. Remote resolution experiments

7. Related work

Recent work on FI resolution in distributed platforms adopts a two-level approach to describe features, functional and resolution [18]. Both levels require programming skills, which should be avoided at the resolution level.

Cooperating FI resolvers have been proposed recently, with FIMA-Feature Interaction Management Agents to coordinate the resolution operations [19]. In case of failure of one or more FIMAs, the resolution is compromised. In our approach, in case a node Advisor fails, the Email deliver is not compromise (at the expense of FI occurrence).

8. Conclusions and future work

Our approach respects the main goals and provides additional advantages as:

- It's possible to add new FI cases in Adviser without source code change and no interruption of FI resolution;
- Interoperable with other kinds of Advisers, respecting the same protocol (XML);

- The adviser causes no major degradation to the application performance;

Our implementation requires the Advisor to be installed in the Email server node. Currently, we are working on a James implementation of Advisor installed in remote nodes.

We intend to extend our experimentation with HTTP servers, taking advantage of our previous experience on the Email server.

Finally, we also have perspectives to work on a more powerful logic model, where formulas can be evaluated based on the consequences of other deductions.

Acknowledgment

We would like to thank to "Coopération pour la Science et la Technologie" of France Embassy in Portugal, for all support.

References

[1] L. Blair et al., "A Feature Manager Approach to the Analysis of Component-Interactions", *5th Int'l Conference on Formal Methods for Open Object-based Distributed Systems*, Enschede, The Netherlands, 2002, pp 233-248.

[2] T.F Bowen et al., "The Feature Interaction Problem in Telecommunication Systems", *7th Int'l Conference on Software Engineering for Telecommunication Systems*, 1989, pp 59-62.

[3] R.J. Hall, "Feature Interactions in Electronic Mail", *6th Int'l Workshop on Feature Interactions in Telecommunication and Software Systems*, Glasgow, Scotland, 2000, pp 67-82.

[4] J. Lennox and H. Schulzrinne, "Feature Interaction in Internet Telephony", *6th Int'l Workshop on Feature Interactions in Telecommunication and Software Systems*, Glasgow, Scotland, 2000, pp 38-50.

[5] M. Weiss, "Feature Interactions in Web Services", *7th Int'l Workshop on Feature Interactions in Telecommunication and Software Systems*, Ottawa, Canada, 2003, pp 149-156.

[6] R.G. Crespo, L. Logrippo and Gray,T., "Feature Execution Trees and Interactions", *Int'l Conference on Parallel and Distributed Processing Techniques and Applications*, Las Vegas NV, 2002, pp 1230-1236.

[7] B. MacCarty, *RedHat Linux Firewalls*, Addison-Wesley. Reading, USA, 2003

[8] A. G Hamilton, *Logic for Mathematician*, Cambridge University Press, 1988.

[9] F. M. Carvalho and R.G. Crespo, "An Experimental Distributed Resolution of WWW Interactions", Conference WWW/Internet 2005, Lisbon, Portugal, 2005, pp 283-290.

[10] A. W. Appel, *Modern Compiler Implementation in Java*. Cambridge University Press, UK , 1998

[11] W. Grosso, Java RMI. O'Reilly. Sebastopol, USA, 2001

[12] A.S. Tanenbaum, Modern Operating Systems 2nd edition. Prentice-Hall, Englewood Cliffs, USA, 2001

[13] F. Yergeau, et al., Extensible Markup Language (XML) 1.0 3rd edition, W3C, 2004

[14] B. McLaughlin, *Java & XML Data Binding*, O'Reilly. Sebastopol, USA, 2002

[15] J.B. Postel, Simple Mail Transfer Protocol RFC821, IETF, 1982.

[16] J. Myers and M. Rose, Post Office Protocol - Version 3 RFC 1939, IETF, 1996.

[17] M. Crispin, Internet Message Access Protocol RFC 2060, IETF, 1996.

[18] J. Pang and Blair, L., "Separating Concerns from Distributed Feature Components", Electronic Notes in Theoretical Computer Science 82(5), 2003.

[19] Chentouf, A. et al, "Experimenting with Feature Interaction Management in SIP Environment", Telecommunication Systems: 24(2), 2003, pp 251-274.