The Negotiation Analysis Pattern

Haitham S. Hamza[1], and Mohamed E. Fayad[2]
[1]*Computer Science and Engineering Dept., University of Nebraska-Lincoln*
Lincoln, NE 68588, USA
Ph: +1 402 4729492
hhamza@cse.unl.edu

[2]*Computer Engineering Dept., College of Engineering, San José State University*
One Washington Square, San José, CA 95192-0180
Ph: +1 408 924-7364, Fax: +1 408 924-4153
m.fayad@sjsu.edu

ABSTRACT

Negotiation is a general concept that has wide range of applications that span various contexts. This paper introduces the Negotiation analysis pattern. This pattern aims to provide a model that analyzes the core concept of the negotiation. In order to achieve this goal, Negotiation pattern is built based on the concepts of Stable Analysis Patterns we have introduced before in [2, 3,4]. The paper provides detailed documentation of the proposed pattern. In addition, it demonstrates the usage of the pattern through the mean of examples.

1. INTRODUCTION

Analysis patterns are conceptual models that model the knowledge of the problem domain. They aid developers in understanding the problem rather than showing how to design a solution. Developing conceptual models for recurring problems can reduce cost, time, and errors in analysis phase.
In our every day life, there are various situations where negotiation usually can place. For instance, buying or selling usually involves some sort of negotiation (e.g. buying or selling a home or a car). In software systems, negotiation appears frequently in the development of different applications. For instance, developing software for online auctions and shopping might involve the negotiation of the price and/or the negotiation of different product aspects.

More technically, negotiation is an essential part in the development of next generation Web-based devices and appliances. Today, devices that need to access the Web diverge greatly in their capabilities, making it highly desirable for the same resource to be available in several different representations (different languages for example). Negotiation algorithms play a fundamental role in aiding servers to decide which representation of a document a device should be given. In this case, the browser (or the client agent) will indicate its preferences by including a header in the request.

In this paper, we present an analysis pattern that models the core aspects of the Negotiation concept. The pattern is documented using the following elements: Problem, Forces, pattern structure and participants, examples, and consequences.

2. PROBLEM

The fact that negotiation concept does span a wide range of spectrum of heterogeneous applications, along with the fact that the negotiation concept itself does not change whenever it appears, both makes the development of a model that captures the core knowledge of the negotiation concept both desired and challenging. Developing such generic and accurate model is not easy and this leads to the main question: How can we build a negotiation model that can be used to model the negotiation problem in any application?

3. Forces

We differentiate between two kinds of forces: *Developmental forces* and *Usage forces.* Developmental forces reflect the conflicting issues that we encounter while trying to capture and present the common aspects of different negotiation systems. The developmental forces share most of the common complications that appear in the realm of domain engineering and modeling, in particular, when performing commonalties and variabilities analysis for a given domain. In the context of this paper, we can consider Negotiation as the domain that we want to analyze. Developmental forces can be confronted by using similar techniques similar to those proposed for domain analysis. However, in this paper, we use the concepts of software stability [7], and stable analysis patterns [2,3,4] to accomplish this task (A brief description of these concepts is presented in the Solution Section below). On the other hand, Usage forces are similar to the traditional notion of forces that is used in conventional pattern documentation. These forces reflect complications that the user of the pattern may encounter in her system and might be solved by the proposed pattern. For instance, even for a single system that require negotiation as a component, the developer may think of how to model different parties involved in the negotiation, or how to design the system so it can be modified in a future to accommodate a foreseen requirement. In the following, we summarize some of the developmental and usage forces in negotiation.

Negotiation pattern should resolve the following forces:

3.1 Developmental Forces:

- Negotiation spans many applications that are completely different in their natures.
- The Negotiation entity can be an organization consisting of many persons, each having a role in the negotiation process. For instance, there can be one individual who is responsible for negotiating financial issues, another who is responsible for negotiating issues related to management, and so on. Therefore, our pattern should be flexible enough to handle different negotiator structures.

- The ultimate goal of any negotiation is to reach an agreement between the negotiators. However, the nature of this agreement varies tremendously from one application to another and from one context to another. An agreement that might be reached while negotiating a political conflict is completely different from that reached while buying a car. Therefore, the pattern should be able to handle these wide variations.

3.2 Usage Forces:

- Your application may require a negotiation between two or more persons, between persons and organizations, or between two non-human entities; in each case, the negotiator structure is completely different. You want your application to have the flexibility to handle all these possibilities.
- Your application may perform negotiation over specific kind of media. For example, your application use mail as a means of conducting the negotiation process; however, you want your application to be flexible to accommodate possible other medias such as email, telephone, or any other kind of medias.
- Negotiation can be performed on one or more affairs at the same time. For instance, negotiation that takes place in the buying and selling context usually involves more than one subject to be negotiated. For example, in buying a car, one can negotiate the price, the warranty, and so on.

4. PATTERN STRUCTURE AND PARTICIPANTS

One approach to tackle the developmental forces discussed in Section 3.1 is by using the concepts of software stability [6]. Software stability stratifies the classes of the system into three layers: the Enduring Business Themes (EBTs) layer, the Business Objects (BOs) layer, and the Industrial Objects (IOs) layer. Each class in the system model is classified into one of these three layers according to its nature. Figure 1 and 2 depict the three layers of software stability model and their relationships.

EBTs are the classes that present the enduring and basic knowledge of the underlying industry or business. Therefore, they are extremely stable and form the nucleus of the SSM. BOs are the classes that map the EBTs of the system into more concrete objects. BOs are tangible and externally stable, but they are internally adaptable. IOs are the classes that map the BOs of the system into physical objects. For instance, the BO "Agreement" can be mapped in real life as a physical "Contract", which is an IO. The detailed characteristics of EBTs, BOs, and IOs and useful heuristics and examples of identifying these concepts in real applications can be found in [5,7,8].

By applying stability concepts to the notion of analysis patterns we introduced the concept of Stable Analysis Patterns [2,3,4]. The idea behind stable analysis patterns is tanalyze the problem under consideration in terms of its EBTs and the BOs with the goal of increased stability and broader reuse. By analyzing the problem in terms of its EBTs

and the BOs, the resultant pattern models the core knowledge of the problem. The goal of this concept is stability. As a result, these stable patterns can be used to model the same problem whenever it appears.

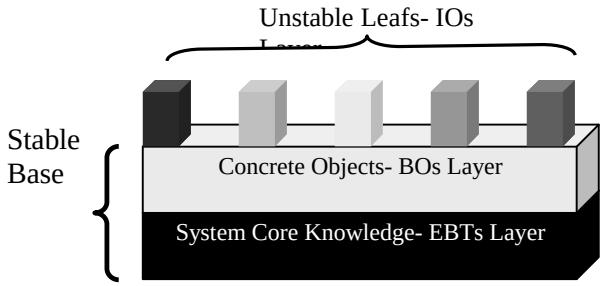

Figure 1. Software stability concepts layout

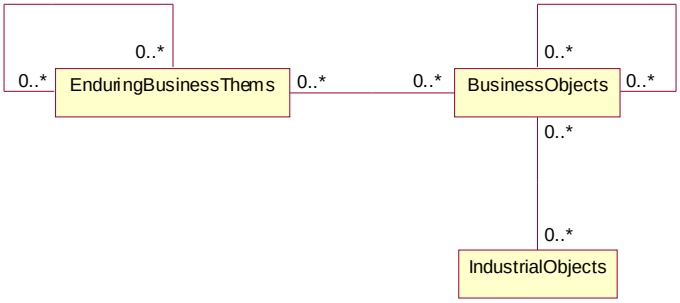

Figure 2. The relation between SSM elements

The proposed solution is to focus on the concept of negotiation trying to extract the main components of the negotiation concept, leaving other domain-specific and/or application-specific components away form this core model. The basic components are represented in generic way that allows the developer to utilize them according to the needs of his/her applications. Figure 3 shows the stable object model of the *Negotiation* pattern.

One way to view the above model is to think of it as a generic model that captures the core requirement of any negotiation process. We differentiate between two main participants in the pattern model, classes and patterns. Classes are defined as in any traditional Object-Oriented class diagram. Patterns present a second level of abstraction to the model, where each pattern is by itself another model that contains classes and, in some cases, other patterns. The reader might refer to the appendix to see how the AnyMedia pattern is expressed as a second abstraction level. As shown in Figure 2, beside the tags that indicate whether the element in the object model presents a class or another pattern, we also use the prefix ‘any’ for patterns. For instance, AnyParty is a stand-alone stable pattern that models the party notion and, hence, can be used to model any party in any applications. The detailed structure of this pattern is out of the scope of this paper.

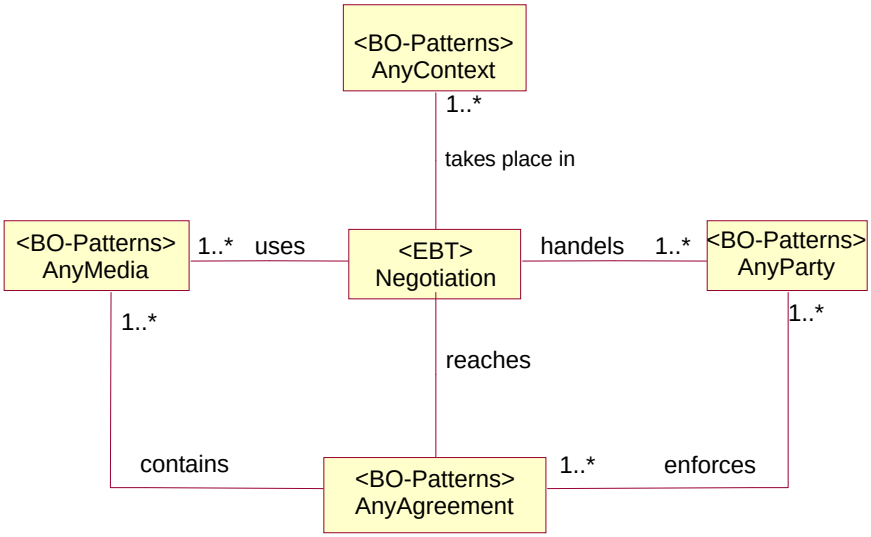

Figure 3. *Negotiation* pattern stable object model

As shown in Figure 3, the *Negotiation* pattern consists of the following participants:

Classes:

- *Negotiation:* Represents the negotiation process itself. This class contains the behaviors and attributes that regulate the actual negotiation process.

Patterns:

- *AnyAgreement:* Represents the result of the negotiation. The ultimate goal of any negotiation is to reach an agreement. Thus, this object presents a core element in any negotiation. It is important to note that in many cases negotiation ends with no agreement and thus it is considered to be failed (the seller of the car did not agree on the price proposed by the buyer and vise versa), however, in this case we expect that the agreement should provide this result by whatever mechanism. So one can view the agreement object as the result of the negotiation, which is not necessary a successful result.
- *AnyParty:* Represents the negotiation handlers. It models all the parties that are involved in the negotiation process. Party can be a person, organization, or a group with specific orientation. The pattern diagram and detailed pattern description is provided in [9].
- *AnyMedia:* Represents the media through which the negotiation will take place. For instance, one can negotiate the price of a good over the phone. Others might use an email or a mail to negotiate specific issues in their business. The pattern diagram and detailed pattern description is provided in [9].
- *AnyContext:* Represents the matters to be negotiated. If we are buying a home, many issues could be negotiated. For instance, the price of the home, the payment procedure, etc. Defining what is the issue to be negotiated is an essential element of

any negotiation process; otherwise, negotiation will have no meaning. The pattern diagram and detailed pattern description is provided in [9].

We shall show in the next Section of this paper the different abstraction levels and some comments on how this abstraction level links to the first abstraction level.

To better understand the roles of each participant in the Negotiation pattern, we present CRC –cards for each participant. The CRC names the class, responsibility, and its collaborations. The CRC card also names a role for each class, which is useful for identifying the class responsibility. Each class should have only one and unique responsibility. The collaboration consists of two parts: clients and server. Clients are classes that collaborate and have relationship with the named class. Server contains all the services that are provided by the named class to its own clients [10].

It is worth to point out that in documenting CRC – cards for stable patterns we deal with any pattern that are included within the main pattern itself as a class. That is, each sub-pattern will be represented by a CRC-card that documents its responsibility and collaborations as a black box. To avoid any confusion, and for simplicity, we do not care about how the sub-pattern handle its reasonability according to its internal structure, all what we care about here is that this sub-pattern will perform the task as a block, leaving the other details to the second abstraction level of the pattern description. For instance, the CRC-cards of the sub-pattern AnyMedia will show the details of each class in the black box AnyMedia.

Negotiation (Negotiation Descriptor)		
Responsibility	**Collaboration**	
	Clients	**Server**
Describes the negotiation rules and regulations to the negotiating parties.	AnyAgreement AnyParty AnyMediaAnyContext	defineRules()

Pattern **:AnyParty** (Negotiation handler)		
Responsibility	**Collaboration**	
Performs and finalizes the negotiation.	**Clients**	**Server**
	Negotiation AnyAgreement	negotiate() approve() agree()

Pattern **:AnyAgreement** (Agreement)-Pattern		
Responsibility	**Collaboration**	
Describes agreement terms and conditions.	**Clients**	**Server**
	-Negotiation AnyParty	option() sign() execute() insert-option()

Pattern **:AnyMedia** (Connector)-Pattern		
Responsibility	**Collaboration**	
Communicates negotiation issues between negotiators.	**Clients**	**Server**
	Negotiation	connectParties() display() illustrate()

Pattern **:AnyContext** (Motivator)-Pattern		
Responsibility	**Collaboration**	
Defines the reason of the negotiation process.	**Clients**	**Server**
	Negotiation	defineBoundary() contextLimit()

5. EXAMPLES

In order to illustrate the use of the *Negotiation* pattern in different applications, two examples are presented: Negotiation of buying a car, and Content Negotiation using Composite Capability/ Preference Profile (CC/PP). Since the objective of these examples is to demonstrate the usage of the proposed pattern, and for simplicity, these examples do not present the complete model for the problem. Instead, they focus on the part that involves the negotiation process. More detailed analysis (use case diagrams, use case descriptions, sequence diagrams, and state transition diagrams) of these two examples can be found in [2] with some modifications.

Example 1: Negotiation for buying a car

In buying a car, a negotiation concerning the car's price and warranty usually takes place. This example models the simple negotiation that might be involved in buying a car. Figure 4 gives the Use Case diagram for this example, and a sample sequence diagram for the Prepare Contract use case is shown in Figure 5. Due to space limitations, we omit the details of the use case model. More details can be found in [2]. Figure 6 shows the stability model of the negotiation used in buying a car. Classes that are not in the original *Negotiation* pattern are colored in gray.

Figure 4. Use Case Diagram for the car example

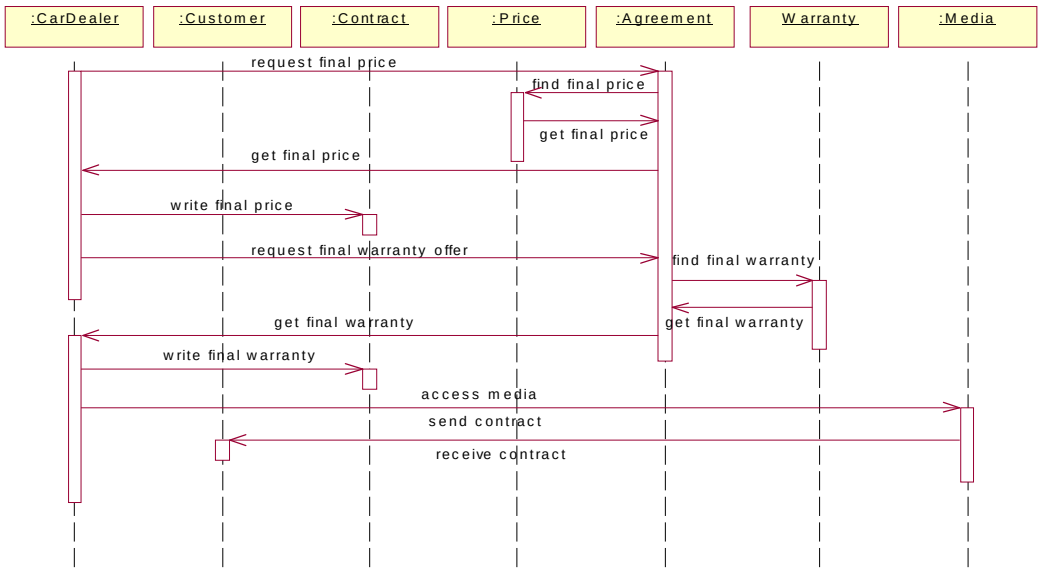

Figure 5. Use Sequence diagram for Prepare Contract use case.

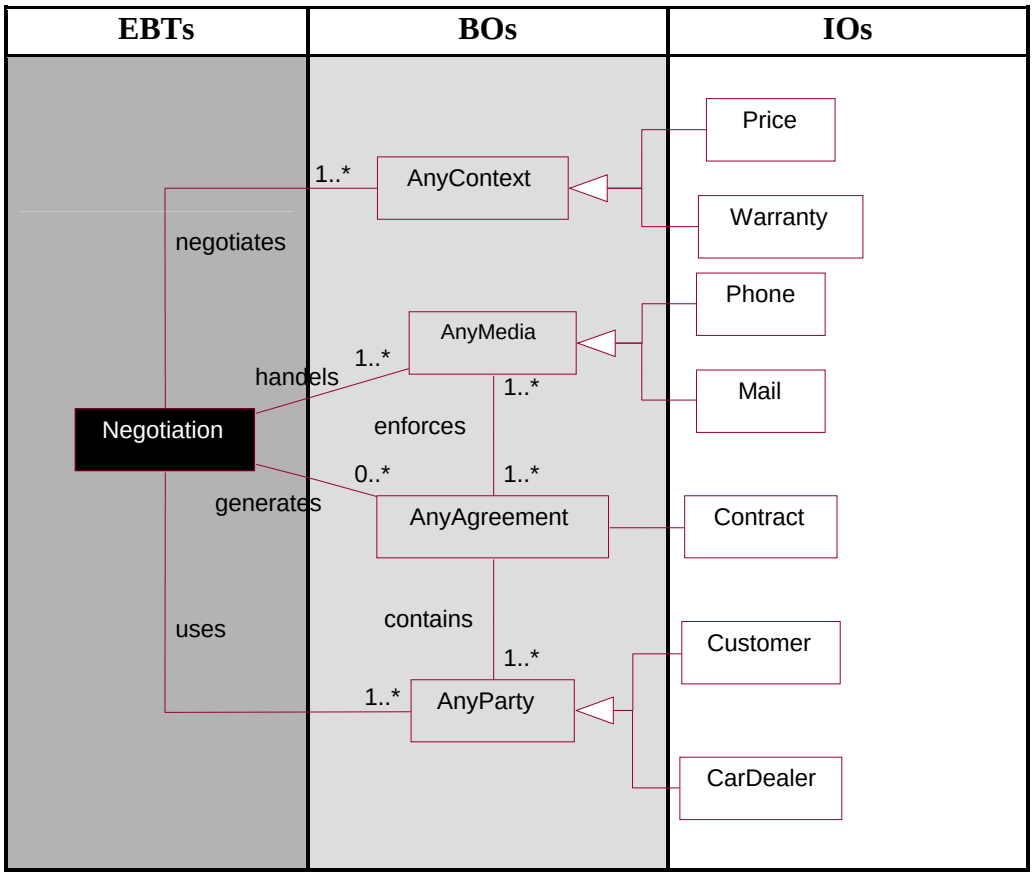

Figure 6. Stability model of the negotiation in buying a car example

Example 2: ***Content Negotiation using Composite Capability/ Preference Profile (CC/PP)***

Today, very heterogeneous devices are required to access the World Wide Web; yet, each device has its own set of capabilities. Therefore, a negotiation between the client and the server should take place in order for the server to know the capabilities of these devices in order to provide the appropriate contents. One possible techniques of performing content negotiation is called *Composite Capability/Preference Profile* (CC/PP) [1]. A possible scenario of CC/PP content negotiation is given in Figure 7. Figure 8 shows the stability model of this example. Again, classes that are not in the original *Negotiation* pattern are colored in white (which are the IOs of the system). The use case model and sample sequence diagrams of this example are given in Appendix B. More details on the dynamics of the and behavior of this example can be found in [2].

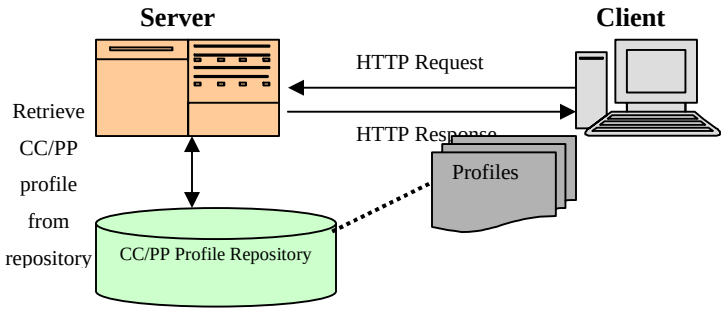

Figure 7. Possible scenario of content negotiation using CC/PP.

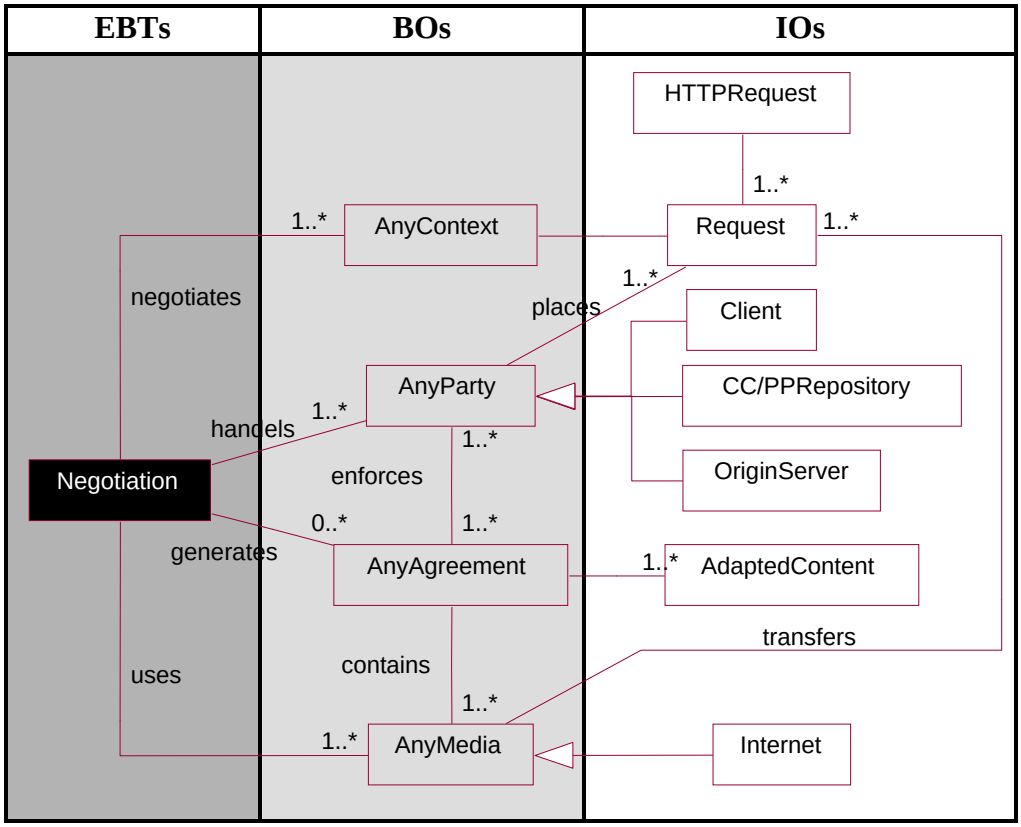

Figure 8. The stability model of the content negotiation example

Abstraction Levels in the Negotiation Pattern

To ease the understanding of the Negotiation pattern, we use different abstraction levels, where each abstraction level provides more details. So far, we have only shown the first abstraction level of the Negotiation pattern (Figure 3). Abstraction levels in the context of stable patterns are obtained through a step-wise decomposition process. For instance, in the Negotiation pattern, the BO called "AnyMedia" is presented as a sub-pattern in the

first abstraction level; however, in the second abstraction level, this black box should be replaced by its detailed structure (See Figure A.1 in the Appendix). In every abstraction level we decompose all the sub-patterns only once. For example, when we perform the first decomposition process on the Negotiation pattern, we will replace the sub-patterns "AnyMeida", "AnyParty", AnyAgreement", and 'AnyContext" by their detailed structures. In summary, to avoid the complication of expressing the overall pattern structure at once, we proceed step by step while replacing sub-patterns of the original pattern. The decomposition process continues until we express the pattern in classes, and hence, we end up with conventional class diagram with no sub-patterns.

The objective of the step-wise decomposition process above is not just to ease the pattern understanding, but also it helps in detecting and dealing with overlapping of different classes and sub-patterns. Stable patterns may contain some classes and sub-patterns that overlap in their roles. For example, consider the very common case of the pattern AnyParty (See Figure A.2 in Appendix for the detailed structure of this pattern) that exist in many patterns. The AnyParty pattern should be presented only once in the detailed structure of the main pattern. For example, in the Negotiation pattern in Figure 3, the sub-patterns "AnyParty" also exist in the structure of the sub-pattern AnyMedia. In this case, the two "AnyParty' sub-patterns should be unified and represented only once.

Nonetheless, to unify overlapping classes while moving through different abstraction levels is not always obvious. A careful examination of the role of each class and sub-pattern must be done. One source of complication in detecting overlapping is when two or more classes or sub-patterns have similar roles but they have different names. In such case, both components should be unified to avoid redundancy and confusion. These issues and other that are related to pattern integration are covered in great details in [9], and [11]. Figure 9 illustrates part of the second abstraction level of the Negotiation pattern.

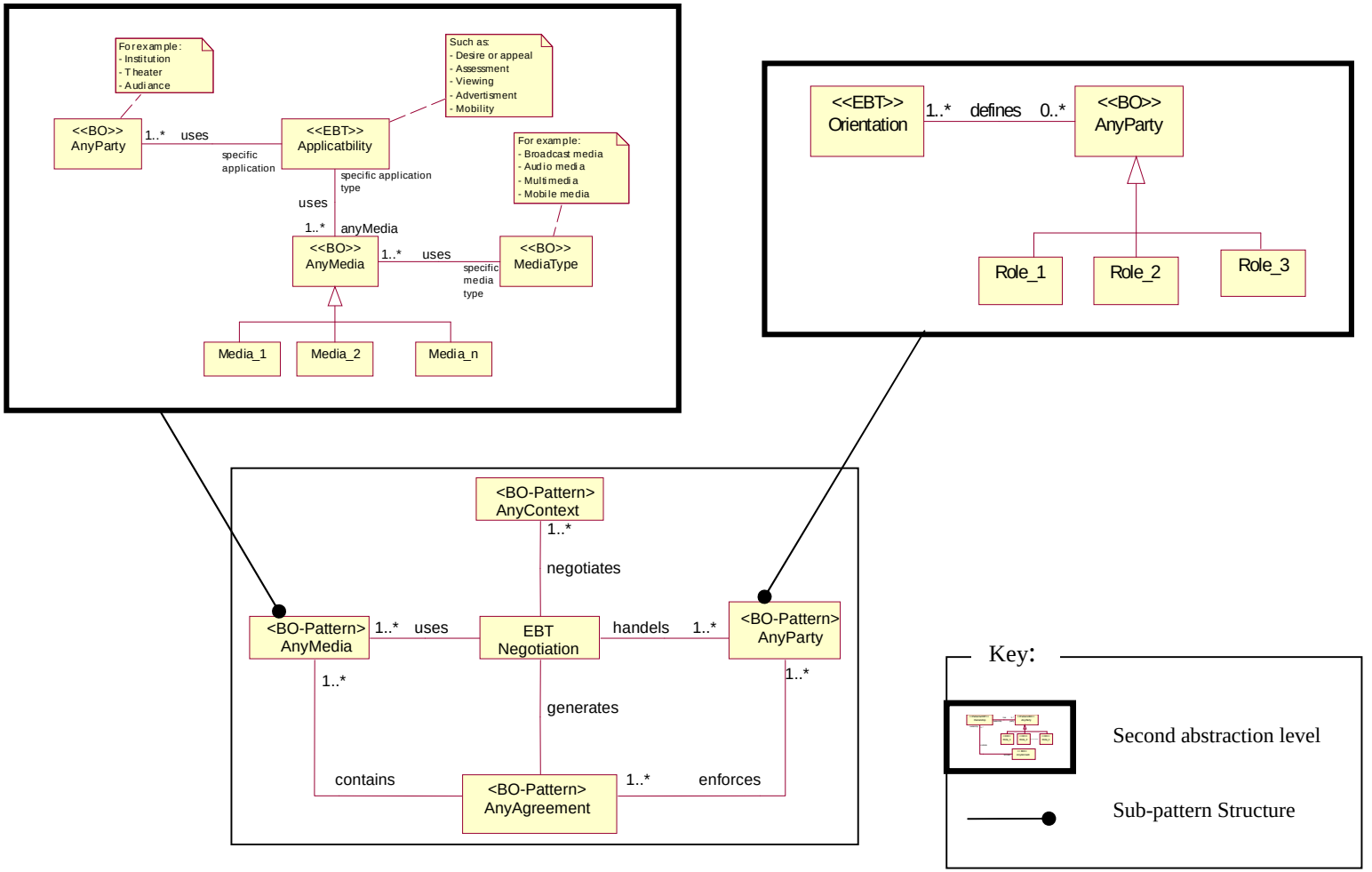

Figure 9. Illustration of different abstraction levels in the Negotiation pattern

6. CONSEQUENCES

The *Negotiation* pattern has the following benefits:

1- *Adaptable for Required Goals.* The Negotiation pattern maintains a high level of adaptability to best achieve the goals of modeling the negotiation notion. For instance, negotiation in buying a car will enforce specific properties that the negotiation process should follow. The determination of the application type will adapt the negotiation pattern to best meet the goals of this negotiation.

2- *Consider Different Media Nature*. The Negotiation pattern considers the negotiation over different media natures. This is achieved by the ***AnyMedia*** pattern, which specifies the media type and its possible kinds.

3-*Easy to Extend.* The Negotiation pattern allows a high level of extensibility to handle more complex applications, or to handle new feature within the same application.

The *Negotiation* pattern has the following limitations:

1- Hidden Patterns Description. At first glance, it would be hard to capture all of the underlying issues that are related to the different patterns that composite the main *Negotiation* pattern. For instance, detailed agreement issues are not seen in the main pattern; however, these issues should be considered within the AnyAgreement pattern details, which therefore, present a second level of details for the original pattern.

2- No Industrial Objects to Clarify Pattern Applicability. In general, understanding conceptual models is not an easy task. In the presented Negotiation pattern there are no IOs attached to the pattern itself, which makes the pattern's applicability not very obvious from just reading its structure. However, attaching such IOs (which are implementation details, and usually application-dependent objects) will narrow the applicability of the pattern. Showing detailed examples for the pattern applicability makes the pattern usage obvious; yet, preserve the generality of the main pattern. It worth to point out that one important feature of the stable patterns conceptual models is that objects' name are carefully chosen to be both general, yet easy to understand. For instance, even though the Negotiation pattern is a conceptual model; yet, all its objects have clear names that make the pattern easy to visualize to some extent even before reading through the provided examples.

7. CONCLUDED REMARKS

In this paper we have proposed a conceptual analysis pattern for negotiation. Our main objective was to provide a pattern that can accurately capture the core aspects of the negotiation concept, while at the same time, can maintain an appropriate abstraction level that makes it easy to be integrated into any application.

Each object in the *Negotiation* pattern has its clear role independent of the application that this pattern will be used in. For instance, *AnyMedia* as an object exists and has the same role whenever we need to negotiate; however, the kind of this media will be determined based on the specific application that uses the *Negotiation* pattern.

The developed analysis pattern has been used to model two problems with different natures. We found that the proposed pattern is flexible enough to be easily incorporated into the two developed applications.

Acknowledgement

The authors would like to thank Ivan Araujo, our shepherd, for his faithful help and valuable discussions and comments that have improved the presentation and the content of the paper considerably.

We also would like to thank Till Schümmer for his great effort and discussions during the shepherding of the paper in EuroPLoP 03.

REFERENCES

[1] Composite Capability/Preference Profiles (CC/PP): A user side framework for content negotiation. W3C Note 21 July 2000. http://www.w3.org/TR/NOTE-CC.

[2] H. Hamza "A Foundation For Building Stable Analysis Patterns." Master thesis. University of Nebraska-Lincoln, 2002.

[3] H. Hamza. "Building Stable Analysis Patterns Using Software Stability." 4th European GCSE Young Researchers Workshop 2002 (GCSE/NoDE YRW 2002), October 2002, Erfurt, Germany.

[4] H. Hamza and M.E. Fayad. "Model-based Software Reuse Using Stable Analysis Patterns" ECOOP 2002, Workshop on Model-based Software Reuse, June 2002, Malaga, Spain.

[5] H. Hamza and M.E. Fayad. "A Pattern Language for Building Stable Analysis Patterns", *9th Conference on Pattern Language of Programs (PLoP 02)*, Illinois, USA, September 2002.

[6] M.E. Fayad, and A. Altman. "Introduction to Software Stability." Communications of the ACM, Vol. 44, No. 9, September 2001.

[7] M.E Fayad. "Accomplishing Software Stability." Communications of the ACM, Vol. 45, No. 1, January 2002,

[8] M.E. Fayad. "How to Deal with Software Stability." Communications of the ACM, Vol. 45, No. 4, April 2002

[9] M.E. Fayad, H. Hamza, and M. Cline. Stable Software Patterns: Analysis, Design, and Applications. (Book in progress)

[10] M.E. Fayad, H.S. Hamza, and H.A. Sánchez. A Pattern for an Effective Class Responsibility Collaborator (CRC) Cards, The 2003 IEEE International Conference on Information Reuse and Integration, Las Vegas, NV, October 2003.

[11] H. Hamza, and M.E. Fayad, "Stable Analysis Patterns," (*in preparation*).

Appendix A

A. 1 *AnyMedia* Pattern

Problem

How to build a model that can be used to present any media in any application?

Context

The pattern can be used to model any media of any type and any kind. For instance, a media type such as multimedia can be of any kinds (e.g., image, voice, etc.).

Solution and Participants

Figure A.1 shows the object diagram of the *AnyMedia* pattern. The shown model gives the high abstract level of view for the proposed model.

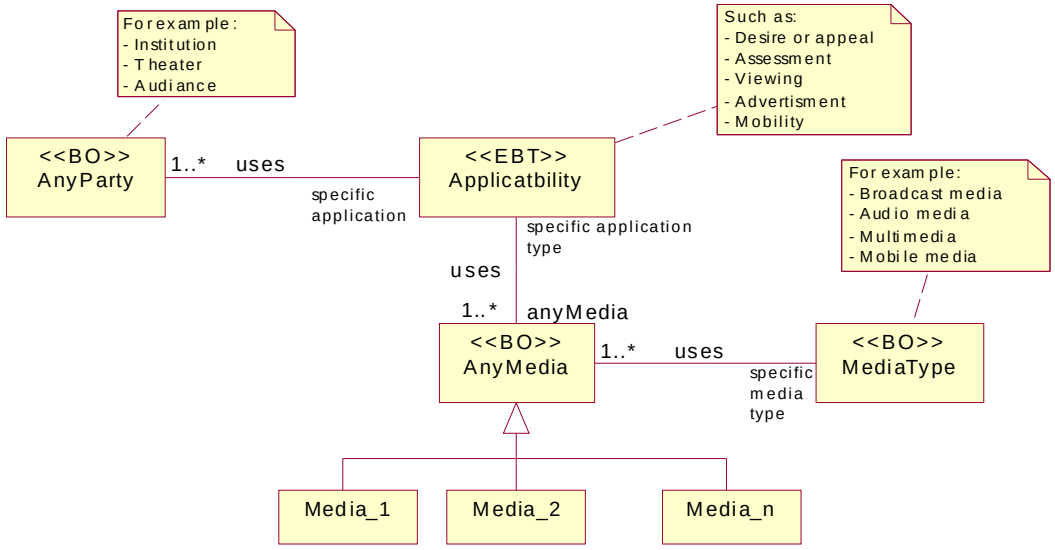

Figure A.1. *AnyMedia* pattern object diagram

Participants

The participants of the *AnyMedia* pattern are:

- *AnyMedia.* Identifies the media to be used
- *MediaType.* Specifies the type of the used media.
- *Applicability.* Describes the purpose of which the media is used.
- *AnyParty.* Represents user of a specific media.

A.2 *AnyParty* Pattern

Problem

How to build a model that can be used to present any party in any application?

Context

The pattern can be used in any application that involves the interaction of any party of any kind and structure.

Solution and Participants

Figure A.2 shows the object diagram of the *AnyParty* pattern.

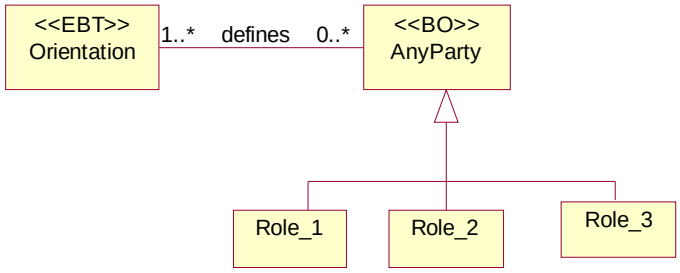

Figure A.2. *AnyParty* pattern object diagram

Participants

The participants of the *AnyParty* pattern are:

- *AnyParty*. Defines the different possible roles within a specific party.
- *Orientation.* Describes the common motive by which party's member are grouped.

Appendix B: Dynamics of the Negotiation Pattern in the Content Negotiation Example

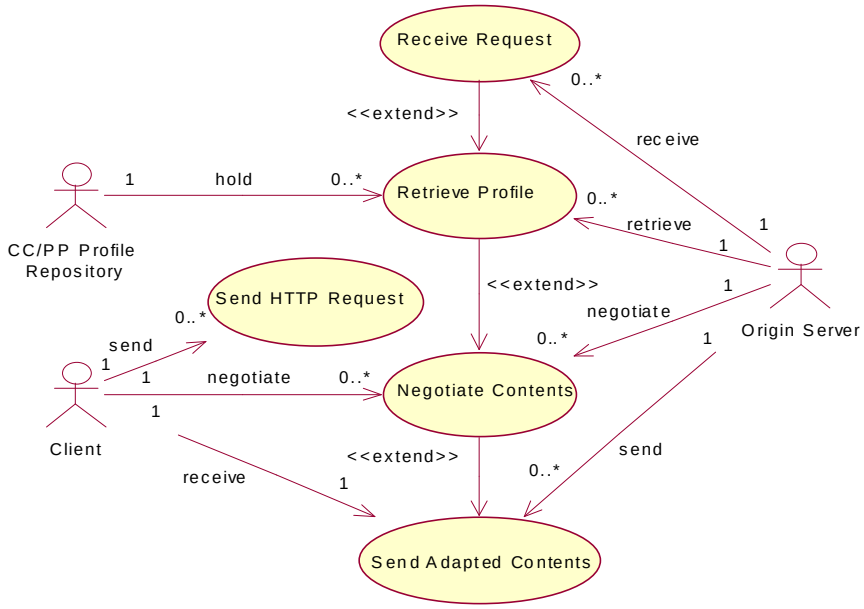

Figure B.1. Use Case Diagram for the Content Negotiation example.

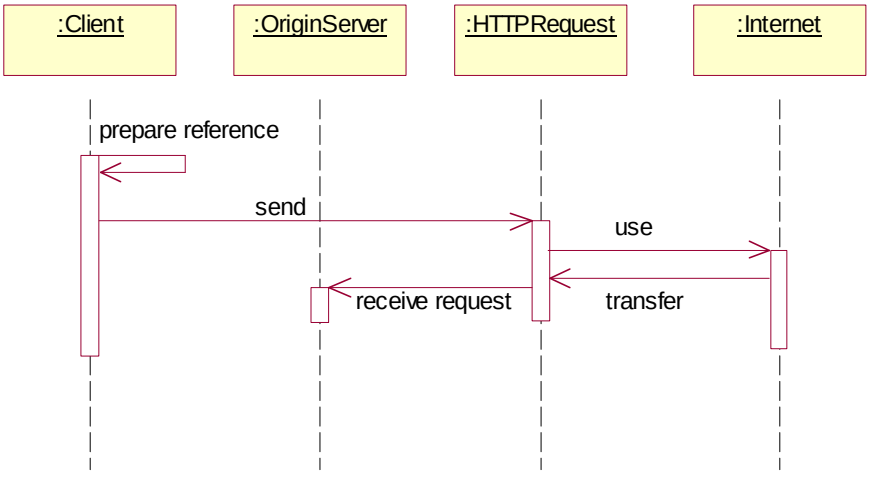

Figure B.2. Send HTTPRequest sequence diagram.

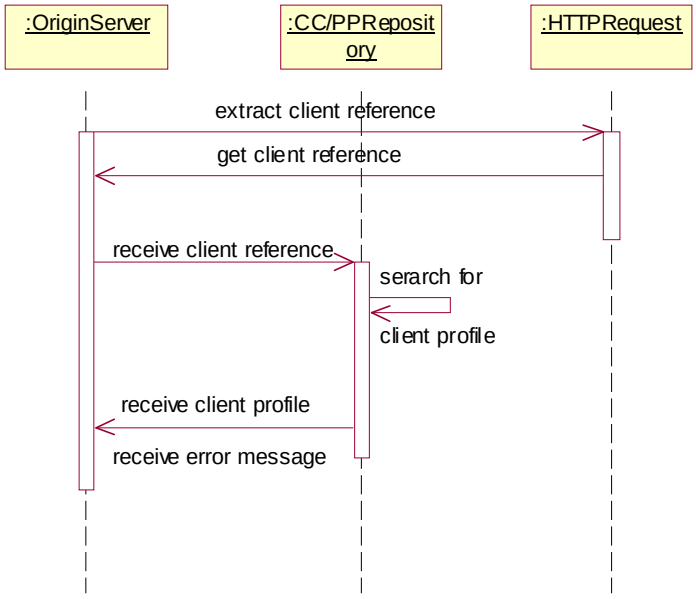

Figure B.3. Retrieve Profile sequence diagram.

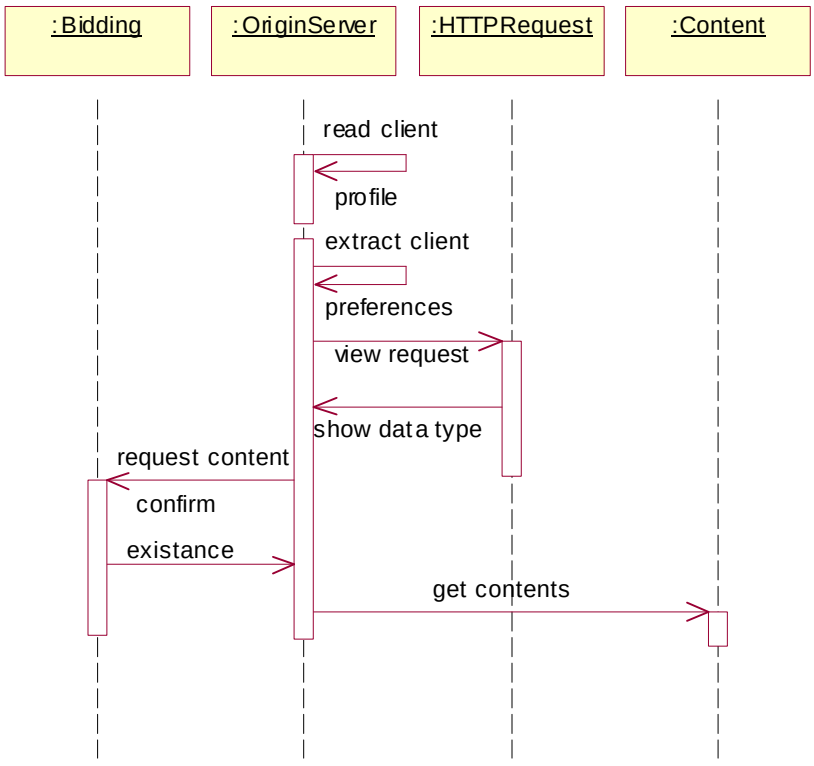

Figure B.4. Adapt Content sequence diagram.

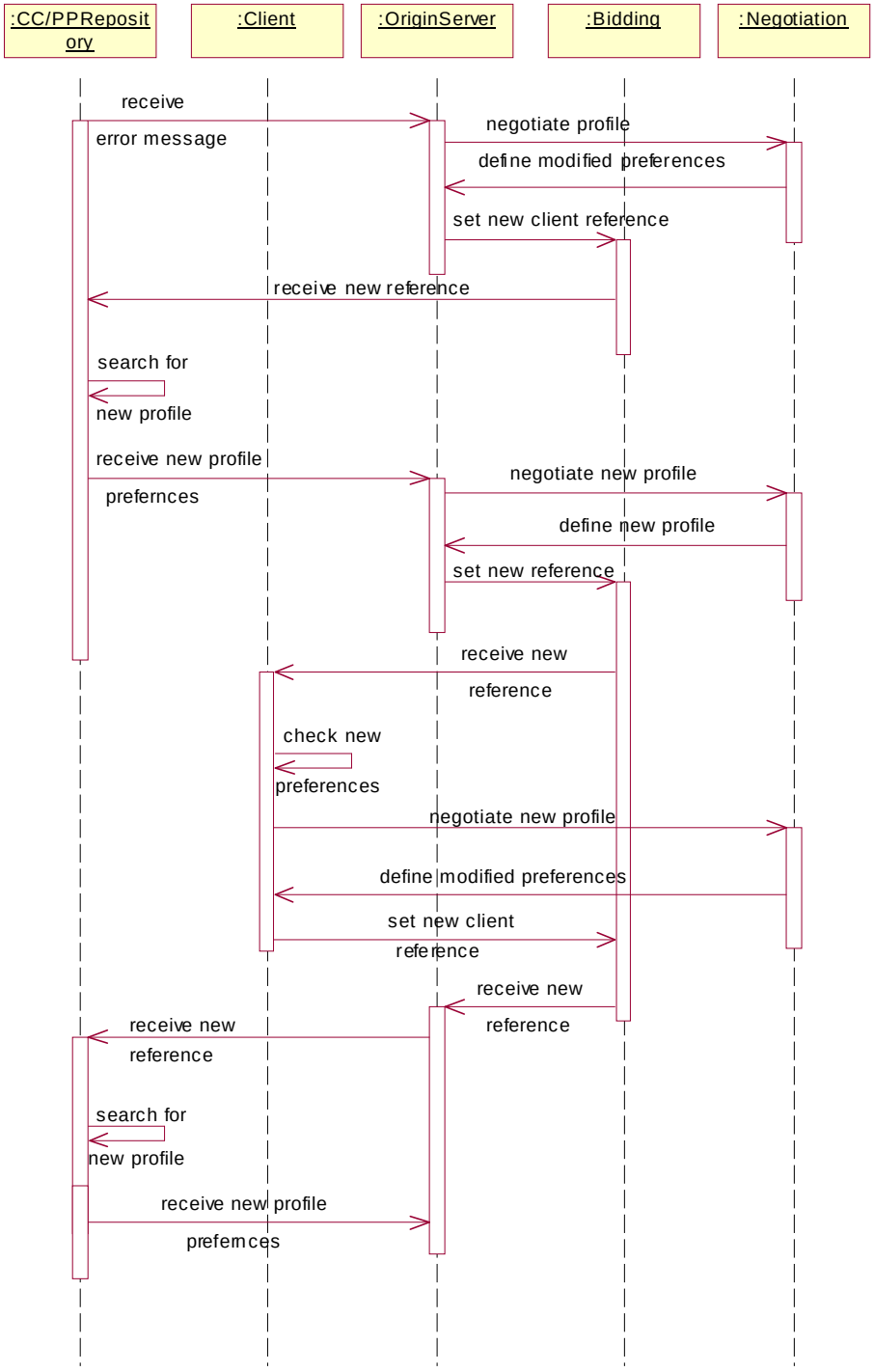

Figure B.5. Negotiate Content sequence diagram.