A Software Stability Model Pattern

Ahmed Mahdy and Mohamed E. Fayad
Computer Science and Engineering Dept.
University of Nebraska-Lincoln
Lincoln, NE 68588, USA
amahdy@cse.unl.edu

Abstract

Building quality software systems has been the focus of many in the field of software engineering. One of the most desirable quality attributes, yet hardest to achieve, is stability [2, 8]. A stable basis provides a foundation for building quality software. The Software Stability approach addresses this issue [6, 7]. The key contribution of this paper is to show how the concepts behind software stability can be used to build stable models.

1. Pattern Name: Software Stability Model (SSM)

This pattern describes how a stable model is built based on the software stability criteria and how to intelligibly present the software stability artifacts.

2. Context

The need for stable models arises in the analysis of problems affected by stability. Model-based reuse is a good example [1]. Building stable architectures is another instance. According to the vision behind introducing software stability, a stable model should be a must for the modeling of any software system [6, 7].

Software Stability Model is usable whenever software stability is used to model a system. This is true because SSM reflects the essential elements of a stable model in terms of the concepts of software stability.

3. Problem

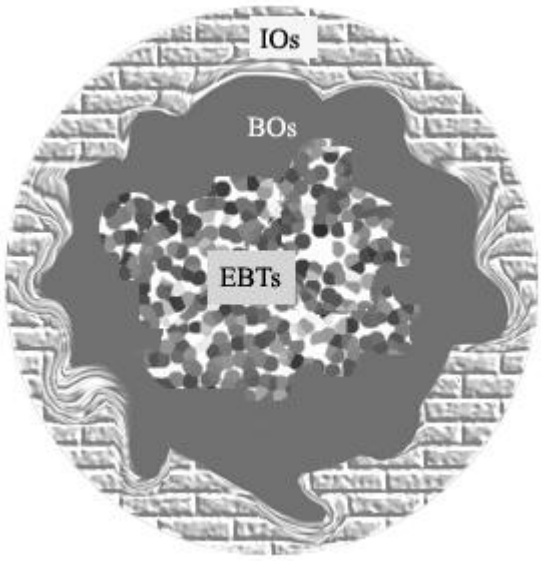
Given a system that needs to be modeled using software stability, how do you translate the components of the system into stability artifacts? A SSM is partitioned into three different levels: Enduring Business Themes (EBTs) [5], Business Objects (BOs), and Industrial Objects (IOs). The EBTs represent elements that remain stable internally and externally. The BOs are objects that are internally adaptable but externally stable, and IOs are the external interface of the system [6].

These artifacts develop a hierarchal order of the system objects, from totally stable at the EBTs level to unstable at the IOs level, through adaptable yet stable at the BOs level. The stable objects of the system are those that do not change over time.

We can also look at the problem from the perspective of tangibility. In addition to the conceptual differences between EBTs and BOs, a BO can be distinguished from an EBT by tangibility. While EBTs are completely intangible concepts, BOs are partially tangible. IOs are tangible objects.

Despite the fact that stability artifacts differ by definition, recognizing them is very arduous. The problem comes from the fact that their definitions are more conceptual than concrete. Moreover, the terms stability and tangibility are hard to visualize, and in many cases are relative. Consequently, people tend to misjudge whether a given element of the system is an EBT, a BO, or an IO. A wrong assignment of an element is directly reflected in the correctness of the model. Figure 1 shows an imaginary picture of how the stability artifacts interlace at the boundaries. SSM aims at resolving the confusion of how to distinguish the EBTs, BOs, and IOs from one another.

Figure 1. Stability Artifacts Interlacing



4. Forces

SSM needs to ensure stability. To achieve this goal, SSM should capture the core knowledge of the system so that a stable core can be built. Among the three artifacts of stability, EBTs and BOs reveal the core knowledge. Consequently, the distinction between the EBTs and BOs from one side and IOs from the other side and between EBTs

and BOs should be maximized in order to avoid drawing a cloudy image of the system core.

The interrelations between the EBTs, BOs, and IOs are hard to realize by nature. This is because the three levels conceptually differ. How an EBT could interact with a BO? Or what is the relation between a BO and an IO? This type of questions needs to be addressed.

Software stability aims at accomplishing system reusability [3]. In fact, the way stable models are built should guarantee reusability [1]. Software stability provides a stable core that can serve applications sharing similar core structure [4]. SSM needs to realize what objects of the stable model to be reused and show how to present them.

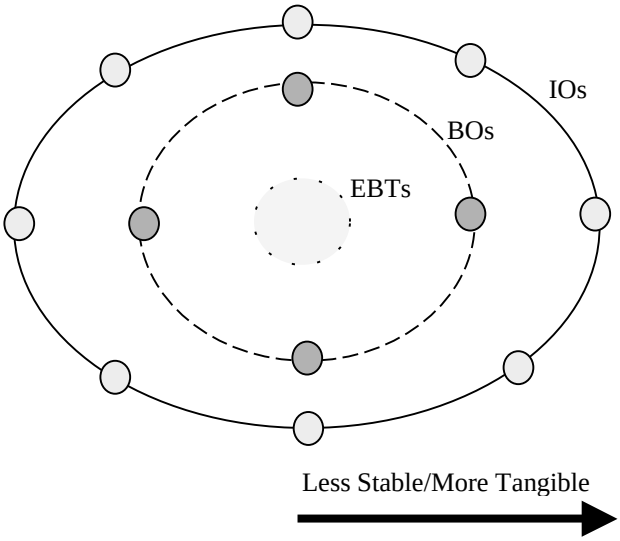
In addition to the stability and reusability characteristics, SSM needs to distinguish the system components according to the tangibility level. SSM has to resolve these characteristics and ensure consistency. A contradiction between any of these characteristics results in a flawed model.

For a system to be reusable, a level of abstraction at which the EBTs, BOs, and IOs are identified has to be configured. Otherwise, the objects of the different levels would interchange positions to result in a non-reusable system. The system would be non-reusable because the modeled core encompasses non-stable elements or/and non-structural components represented by the IOs of the system.

5. Solution

To better visualize SSM, we show first a figure representing the architecture of a stable model according to the stability and tangibility properties of its components. As shown in Figure 2, the EBTs represent the nucleus of the model as they reflect the enduring concepts of the system, while the IOs portray the surface of the system. The BOs lay in between. Intuitively, the further the objects are placed from the interface the more stable and intangible they are.

Figure 2. Stable Models Architecture



5.1 Participants

EBTs

- Represent the enduring themes of the system.
- Comprise the stable and intangible objects of the system.
- Describe the major goals of the systems.

BOs

- Comprise the partially tangible objects.
- Comprise the internally unstable though externally stable objects.

IOs

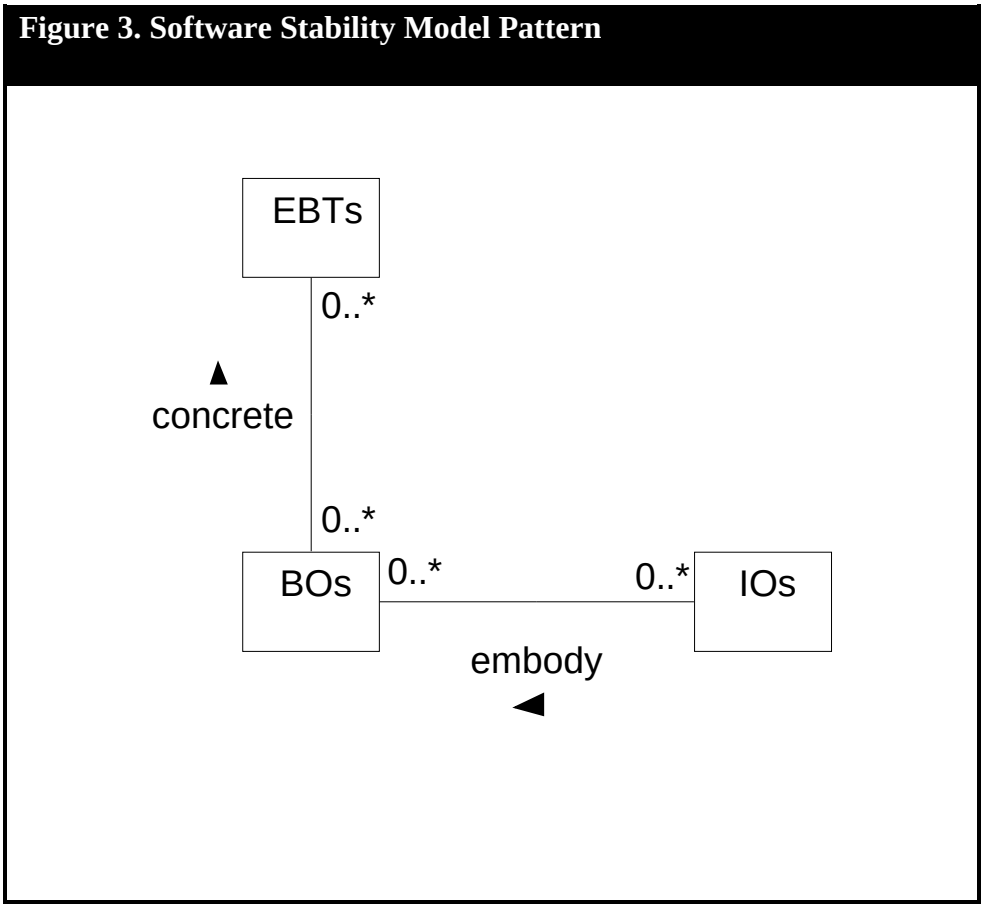
- Depict the surface of the system.
- Represent the physical objects.
- Comprise the unstable objects.

The interactions between the EBTs and BOs form a crucial part of the system. The transition from a conceptual theme to a business-driven object has a great influence on the whole system. An inappropriate transition might diminish one or more of the system objectives, which results in a defective system. These transitions are hard to realize as the EBTs and BOs share the intangibility properties of the system components, the EBTs are fully intangible while the BOs are partially intangible. Such commonality hazes the

distinction between them. On the other hand, direct interactions between the EBTs and IOs are inappropriate. They interact through the BOs. The IOs embody the BOs as they become totally tangible.

5.2 Structure

Figure 3. Software Stability Model Pattern



In addition to the object model, the process of building a stable model can be summarized as follows:

- To find the EBTs, one needs to answer the question "What is this system for?"
- The BOs are the answer to "What is the representation of the intangible conceptual themes into more concrete objects?"
- The IOs answer "What is the physical representation of the BOs?"
- The object model is partitioned into three columns; each column is dedicated to one of the artifacts (i.e. EBTs, BOs, and IOs). Each column encloses the object of its level. This way makes it easier to realize the different components of the system especially in the case of reusing the model to build a new system. Another benefit is avoiding the inappropriate associations among the objects of each level (i.e. an association between an EBT and an IO).

5.3 Collaborations

- The BOs map the conceptual themes of the EBTs into objects.
- The IOs physically represent the BOs.

6. Example

This case models a generic simulator. We follow the SSM process of building stable models. Consequently, we need to answer the three questions in order to identify the EBTs, BOs, and IOs of the system.

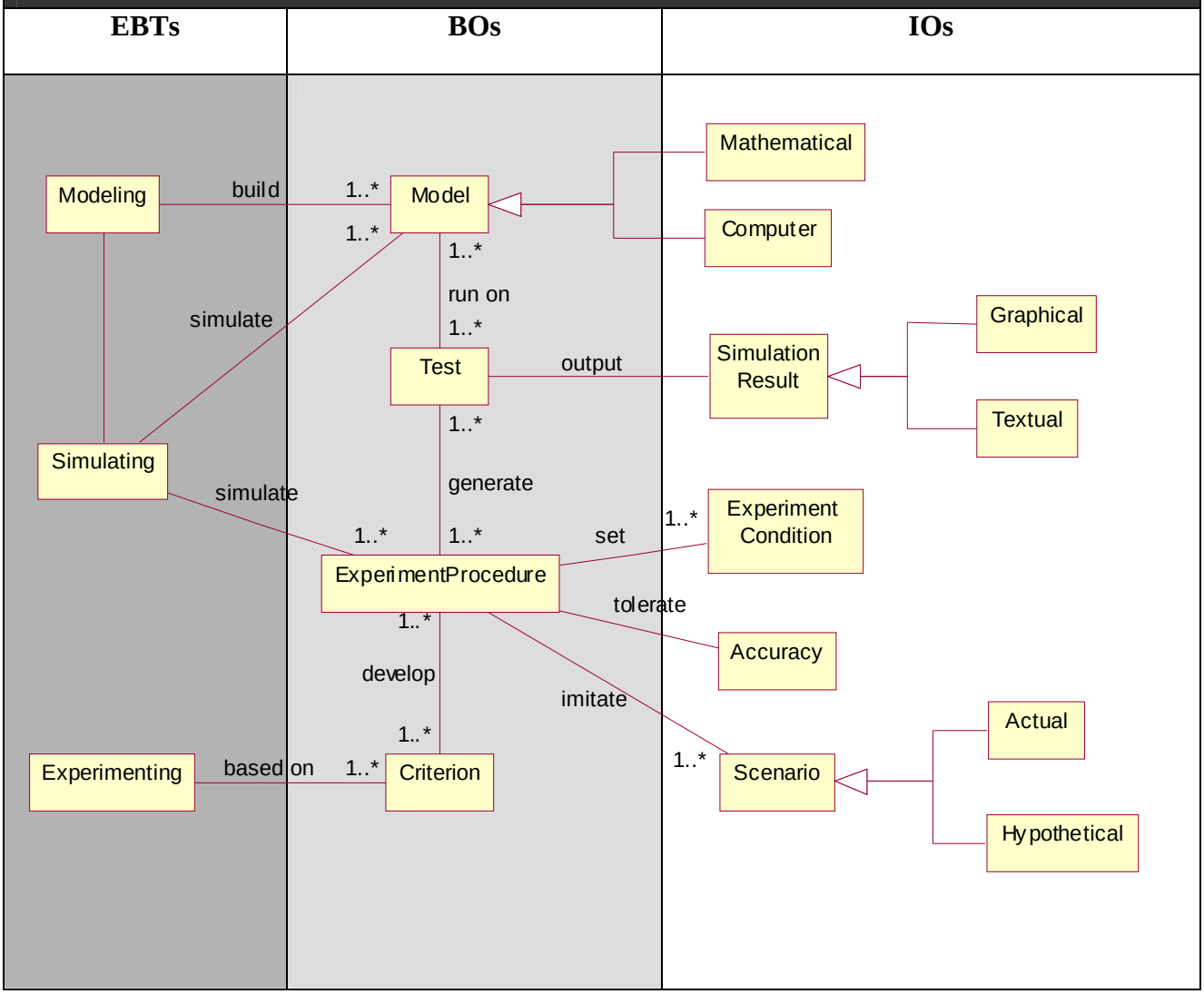
- What is this system for?
  This system is to *model* the chemical reactions behavior in order to *simulate* and/or *experiment* specific instances. Therefore, the EBTs are modeling, simulating, and experimenting.

- What is the representation of the intangible conceptual themes into more concrete objects?
  The modeling theme conceptually refers to a *model*. Experimenting needs some *criterion* to make it alive. To simulate the function of that model according to this criterion, a *test* should follow a *procedure*. Accordingly, the BOs are model, criterion, test, and procedure.

- What is the physical representation of the BOs?
  A model could be either a *mathematical* model or a *computer*-based model. A procedure is represented in terms of a hypothetical or an actual *scenario*, a predetermined *accuracy*, and a set of *conditions*. A test is represented in terms of its *results*, which take the form of graphs or/and text.

Figure 4 shows a stable model of this example.

Figure 4. Simulation Stable Model

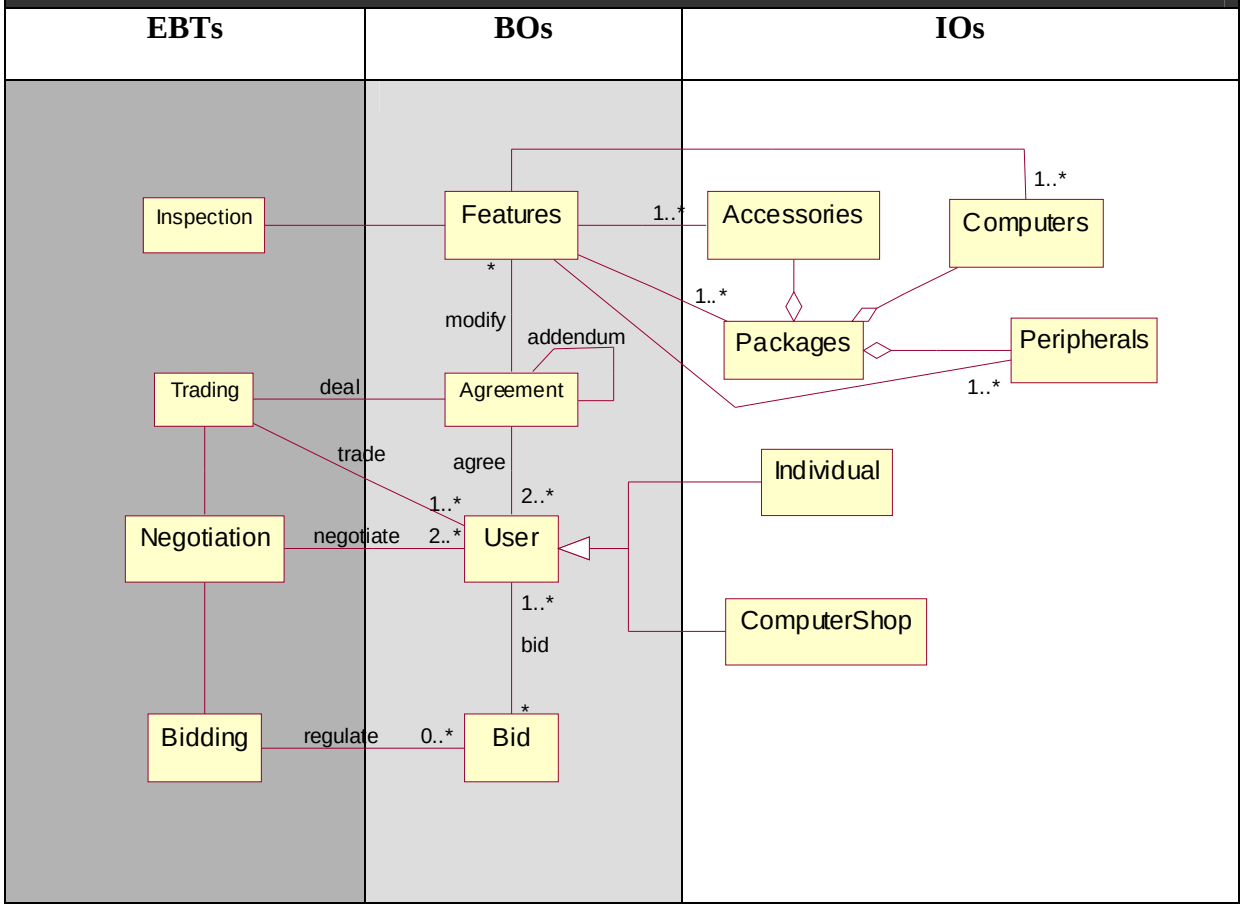


The above example shows how SSM builds a stable model. In the following, we present three case studies to illustrate how SSM insures reusability among applications with similar cores; these case studies are introduced in [1]. The core knowledge is represented by the EBTs and BOs. Thus, systems sharing the core knowledge should share also the EBTs and BOs, and this is what these cases show.

Case I: Computers Trading

This case demonstrates trading computers through a bidding process. One entity specifies its needs; another entity places a bid on such specifications. A negotiation process takes place until a deal/impasse is reached. The process of building a stable model is similar to that of the generic simulator. Figure 5 shows a stable model.

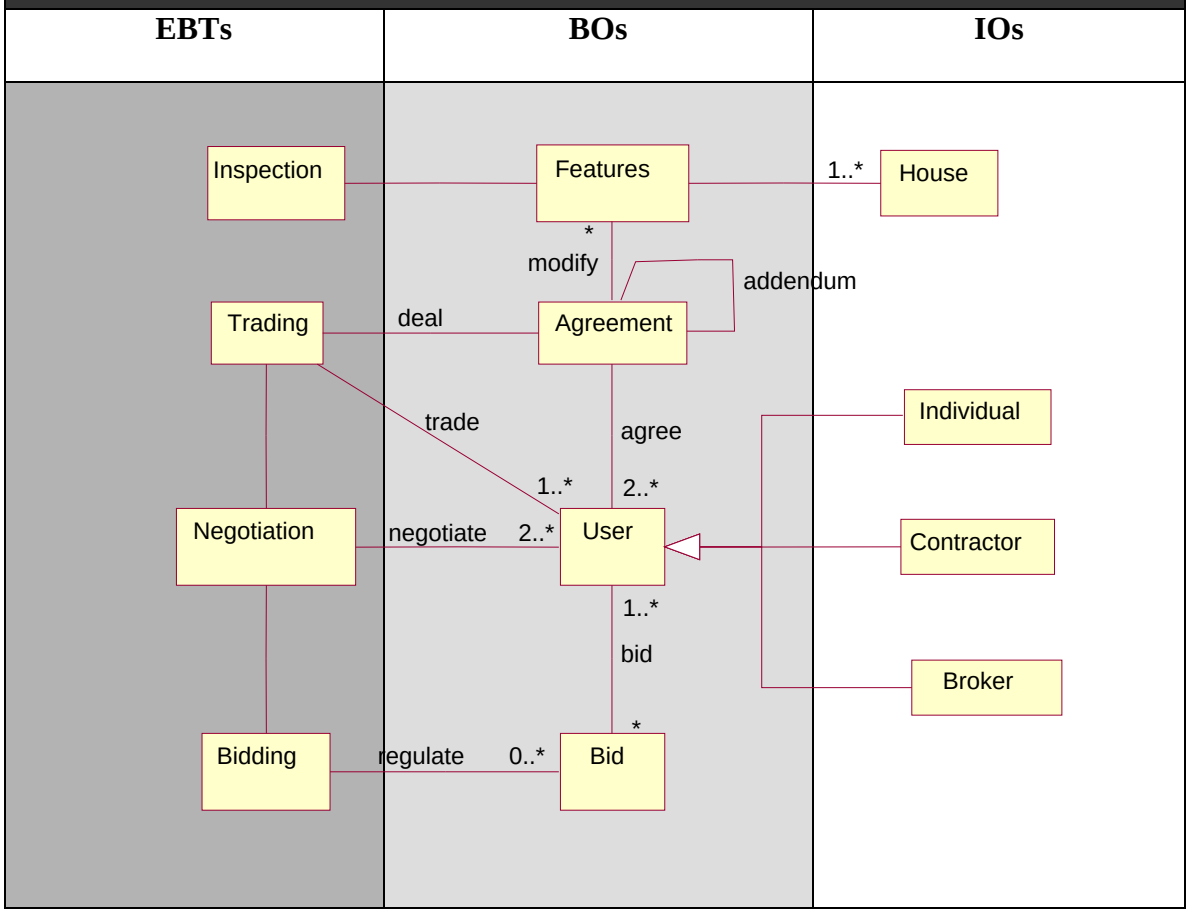
Figure 5. Computers Trading Stable Model



Case II: Buying a House

This case depicts a system for buying houses through bidding too. Case I and Case II look different though buying a house involve bidding, negotiation, trading…etc. In fact, both case studies share the same core. Thus, the answers to the first and second questions of the SSM process are the same. Figure 6 presents a stable model.

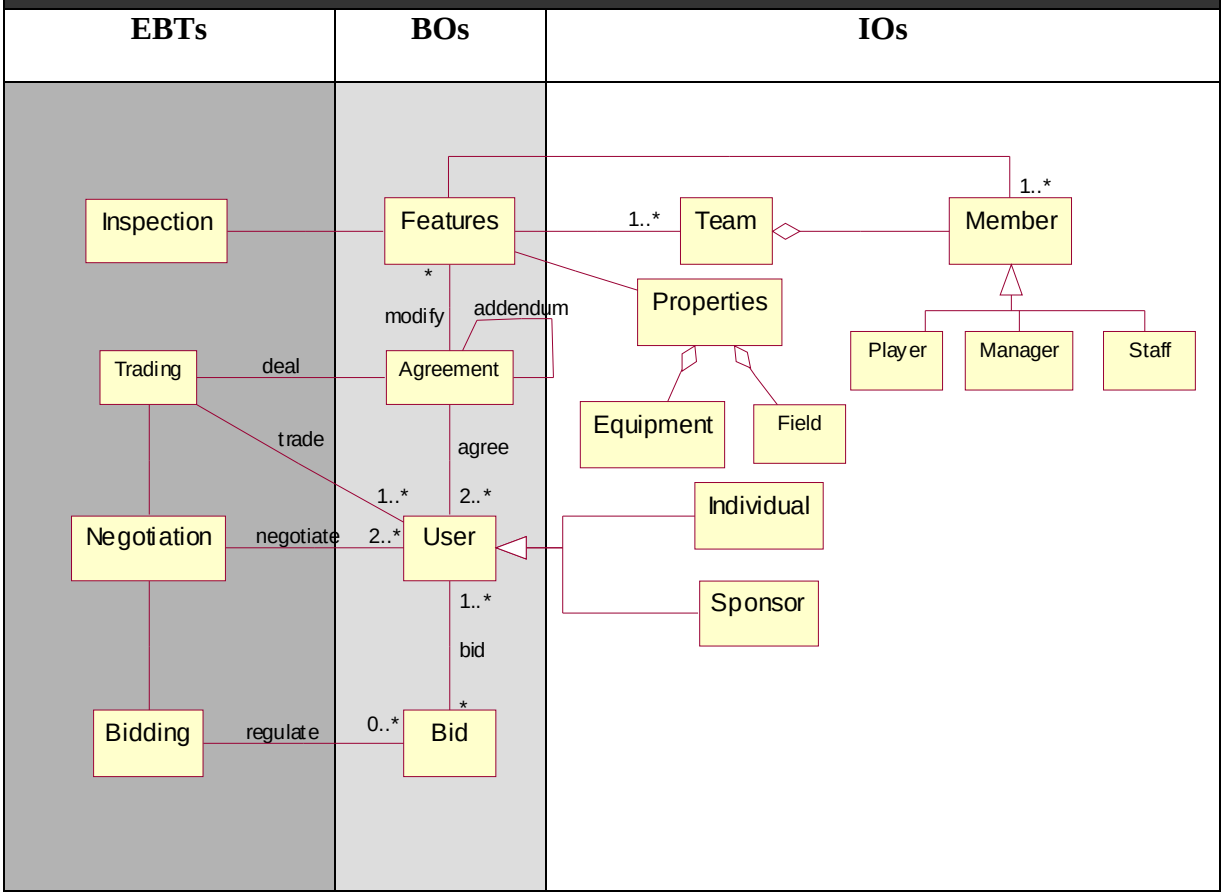
Figure 6. Buying a House Stable Model



Case III: Bidding on a Football Team

Again, this case study appears different from the other two case studies but it shares the core with them. When you buy a football team, you are trading, negotiating, and bidding. So buying a football team is not that different from buying a computer. Thus, the EBTs and BOs are the same. Figure 7 shows a stable model of this case.

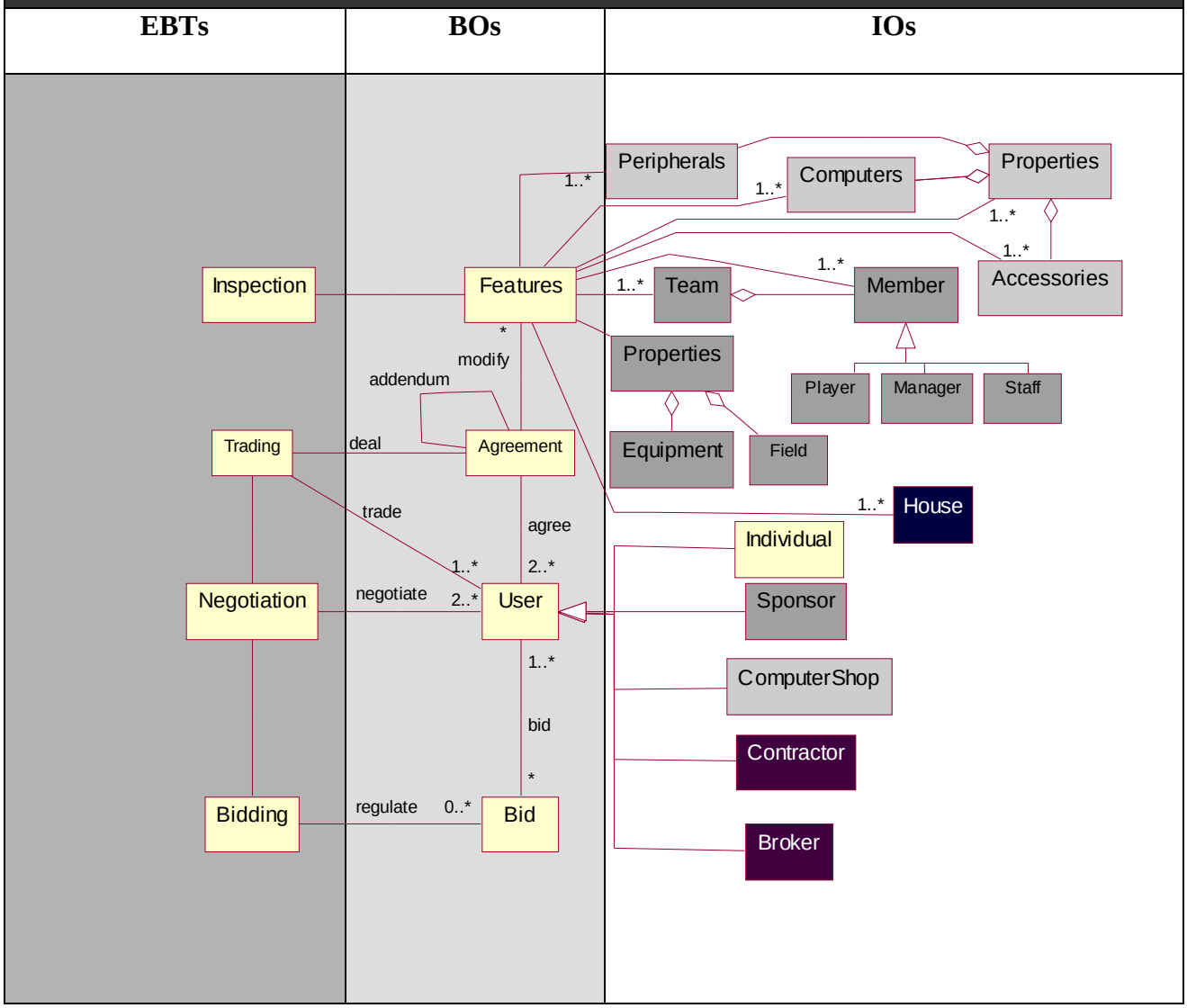
Figure 7. Bidding on a Football Team Stable Model



Although the three case studies deal with different types of trading, the inner core is the same among these applications. Recognizing this fact results from how software stability approaches the problem domain. Identifying the system EBTs directly reflects the goals of the application. If different applications share the same goals, it would be easy to determine their similarities by following the stability approach. Moreover, the BOs will generally be the same. The only differences appear at the IOs level. In our case studies, the three applications have the same EBTs and BOs. Figure 8 plots the three stable models on top of each other to show their similarities.

The way SSM builds stable models assures reusability among common-core applications. The answers to the first and second questions reveal the core knowledge of the system. As a result, the systems with similar cores must have similar answers. Without these questions it is hard to figure out how similar these systems are. As a consequence, SSM maintains the reusability properties of the software stability approach. More important is how the collaborations between the EBTs and BOs recur. The three case studies show how the interrelations remain the same as long as the core remains stable.

Figure 8. Combined Stable Models



7. Consequences

SSM attains stability as it exposes the core knowledge of the system by modeling the EBTs and BOs of the system. Thus, a stable core composed of these artifacts is built. The way SSM identifies the EBTs and BOs through the proposed questions establishes a clear cut between the EBTs and BOs. Hence, confusions between them seldom happen.

SSM ensures the reusability feature of the stable models by building the right core of the system, which is done through the well recognition of the EBTs and BOs. Common-core systems may look different, but in fact, the only differences lay on the surface (i.e. IOs).

If these systems have a common internal structure, it suffices to develop one system for all of these similar applications to be extracted from. At this point, the importance of stable models reusability becomes apparent. The more the systems share, the less will need to be changed [4]. Changes will be made to the IOs, the EBTs and BOs need not be touched. Therefore, the EBTs and BOs can be reused among those applications if they are well recognized.

SSM maintains the consistency between the stability and tangibility of the objects. The tangible objects reflect an essential property of the IOs. According to SSM, an intangible object cannot be an IO, which agrees with the definition of stability. Similarly, the EBTs and BOs are defined in terms of tangibility.

SSM proposes a clear differentiation between the BOs and IOs. The BOs belong to the reusable part of the system, while the IOs are entirely non-reusable. Accordingly, when modeling another system sharing the core knowledge, it is easy to figure out what elements to reuse and what not to. This accomplishes a level of abstraction, which prevents the confusion between the BOs and IOs.

Acknowledgement

We are very grateful to Brad Appleton who shepherded this paper for the help and effort he provided. Thank you, Brad!

References

[1] A. Mahdy, M.E. Fayad, H. Hamza, and P. Tugnawat, "Stable and Reusable Model-Based Architectures" 12th Workshop on Model-based Software Reuse, 16th ECOOP 2002, Malaga, Spain.

[2] D.L. Parnas, "Software Aging", Proceedings of the 16th International Conference on Software Engineering, May 1994, pp 279-287.

[3] G. Arango, "A brief Introduction to Domain Analysis", Proceedings of the ACM Symposium on Applied Computing, April 1994, pp 42-46.

[4] J. Coplien, D. Hoffman, D. Weiss, "Commonality and Variability in Software Engineering", IEEE Software, Vol. 15, No. 6, Nov. 1998, pp 37-45.

[5] M. Cline and M. Girou, "Enduring Business Themes", Communications of the ACM**,** Vol. 43, No. 5, May 2000, pp. 101-106.

[6] M.E. Fayad, "Accomplishing Software Stability", Communications of the ACM**,** Vol. 45, No. 1, January 2001, pp 95-98.

[7] M.E. Fayad, and A. Altman, “Introduction to Software Stability”, Communications of the ACM**,** Vol. 44, No. 9, September 2001, pp 95-98.

[8] R.C. Martin, “Stability”, C++ Report, Feb. 1997.