Towards a Classification of Web Service Feature Interactions

Michael Weiss[1], Babak Esfandiari[2], and Yun Luo[1]

[1] School of Computer Science, Carleton University, Ottawa, Canada
{weiss, yluo}@scs.carleton.ca

[2] Department of Systems and Computer Engineering, Carleton University
babak@sce.carleton.ca

Abstract. Web services promise to allow businesses to adapt rapidly to changes in the business environment, and the needs of different customers. The rapid introduction of new web services into a dynamic business environment can lead to undesirable interactions that negatively impact service quality and user satisfaction. In previous work, we have shown how to model such interactions between web services as feature interactions, and reason about undesirable side-effects of web service composition. In this paper we present the results of subsequent research on a classification of feature interactions among web services. Such a classification is beneficial as we can then search for ways of detecting and resolving each class of feature interaction in a generic manner. To illustrate the interactions we use a fictitious e-commerce scenario.

1 Introduction

Feature interactions are interactions between independently developed features, which can be either intended, or unintended and result in undesirable side-effects. In previous work [9], we have shown how to model undesirable side-effects of web service composition as feature interactions. The formal study of feature interactions is known as the feature interaction problem. This problem has first been investigated in the telecommunications domain [4]. It concerns the coordination of features such that they cooperate towards a desired result at the application level. However, the feature interaction problem is not limited to telecommunications. The phenomenon of undesirable interactions between components of a system can occur in *any* software system that is subject to changes.

Interaction is certainly the very foundation of service-oriented architectures. Web services *must* interact, and useful web services will "emerge" from the interaction of more specialized services. As the number of web services available increases, their interactions will also become more complex. Systems we build will use third-party services, over whose implementation we have little control. Many of the web service interactions will be intended, but others may be unintended and undesirable, and we need to prevent their consequences from occurring. As noted by [7], many of the side-effects are related to security and privacy.

This paper builds on our previous work [9, 11] by providing a categorization of feature interactions in web services by cause, following similar work in the telecommunications domain [2]. We also now propose a unified, realistic, and quite generic case study (the "Amazin" virtual bookstore) that illustrates all the discussed causes while remaining technology-agnostic and easily translatable to other domains. We believe that the case study can be used as a benchmark for future studies in feature interactions in web services.

While in our previous work we had hand-crafted our examples in order to highlight the potential for feature interaction, in this work we used a candid approach in which features were described individually, and without consideration as to their possible participation in feature interactions. The feature interactions that we can observe only arose from composing the services for the scenario in the case study. We believe that this approach strengthens our claims with respect to the pervasiveness of the feature interaction problem in web services.

The paper is organized as follows. We first provide more background on the feature interaction problem as it applies to web services, and on modeling web services as features. We then present our classification of web service feature interactions. To illustrate the interactions we present our case study of a fictitious virtual bookstore. This classification is followed a summary of related work. We conclude with a discussion and an outlook on future research.

2 Feature Interaction Problem

The first generation of web services did not exploit the benefits of a web *of* services. They were either of a simple, non-composite nature (often information services, such as a stock quote lookup service), or provided access to application functionality over pre-existing business relationships. By contrast, the current generation of web services are typically composite (i.e., they are constructed from other, more primitive web services), and offered by third-party service providers, and thus not grounded in existing relationships.

Web services of the first generation were predicated on two implict assumptions: (1) that services developed in isolation would either be used in isolation, or, if part of a composite service, would not interact in inadvertent ways, and (2) that users had full control over the services they used, or there was a common understanding of the operation, and side-effects of these services. We argue that those assumptions are no longer valid for current web services.

Consider the example of a word-processing service that uses two third-party services, spell-checking and formatting [9]. Assume that the user has set her language preference for the word-processing service to UK English. However, let us also assume that, hidden to the word-processing service, the formatting service itself incorporates a spell-checking service. This time, the formatting service does not specify a language preference to the spell checking service. Suppose that the spell checking service uses a US English dictionary by default. The result of the service composition is that the incorrect language option will be applied.

This is a case of an undesirable feature interaction. The concepts of feature and feature interaction originated in the telecommunication domain. A *feature* is the minimum user-visible service unit in this domain [4]. Features are often independently developed and deployed. A *feature interaction* occurs when a feature invokes or influences another feature directly or indirectly. Although many of these interactions are, indeed, intended, undesirable side-effects as a result of the interaction of features are referred to as *feature interaction problem.*

3 Modeling Web Services as Features

Our approach is to model features at the early requirements stage using the User Requirements Notation (URN) [1]. These models allow us to reason about feature interactions, and document detection and resolution strategies. In this approach, the intent and side-effects of a feature are modeled as goals, and their operation in the form of scenarios. The interaction of features is captured in the form of links between goals. Finally, we can represent the allocation of features to subsystems (known as actors or components in URN), and the relationships between these actors. This section provides a brief overview of the approach.

3.1 User Requirements Notation

URN contains two complementary notations: Goal-oriented Requirements Language (GRL) [3], and Use Case Maps (UCM) [8]. In GRL requirements are modeled as goals to be achieved by the design of a system. The main elements of the notation are summarized in Fig. 1. During the analysis, a set of initial goals is iteratively refined into subgoals. These goals and their refinement relationships form a goal graph that shows the influence of goals on each other, and can be analyzed for goal conflicts. The perspectives of different stakeholders

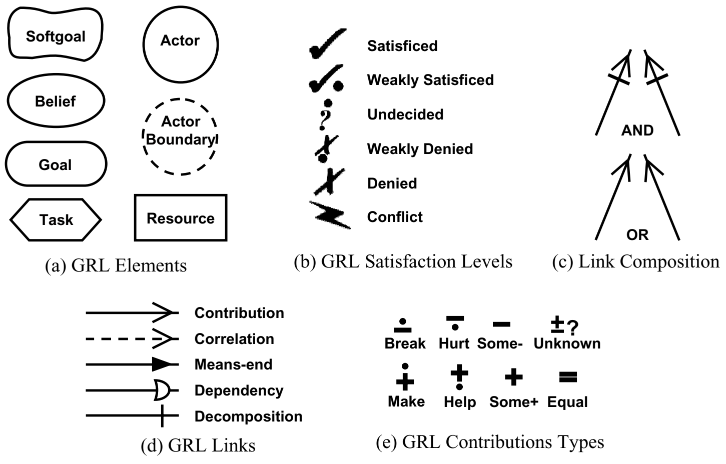

Fig. 1. Summary of the Goal-oriented Requirements Language (GRL)

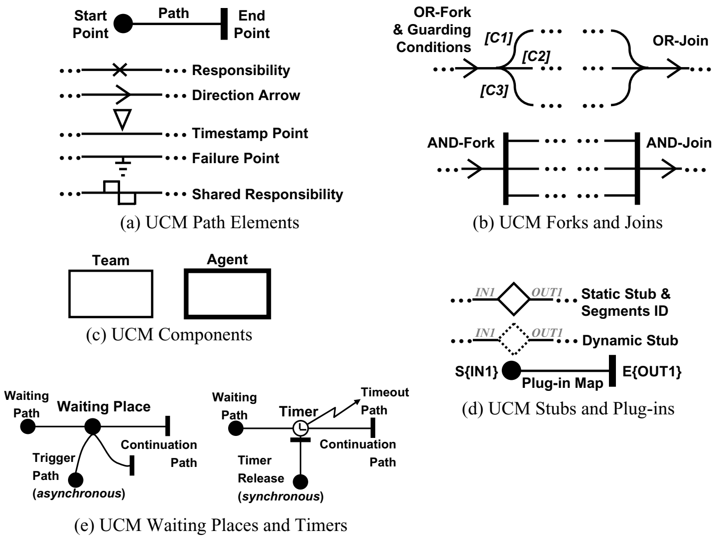

Fig. 2. Summary of the Use Case Map (UCM) notation

can also be described in GRL. For each stakeholder we model their goals, as well as their dependencies on one another to achieve those goals. These goals of one stakeholder can now also compromise the goals of other stakeholders. The objective of the analysis is to determine the design alternative that resolves the goal conflicts in a way that best satisfies all stakeholder's initial goals.

The UCM notation provides a way of describing scenarios without the need to commit to system components. The main elements of the notation are summarized in Fig. 2. A scenario is a causally ordered set of responsibilities that a system performs. Responsibilities can be allocated to components by placing them within the boundaries of that component. This is how we will be modeling feature deployment. With UCMs, different structures suggested by alternatives that were identified in a GRL model can be expressed and evaluated by moving responsibilities from one component (which is the UCM equivalent of a GRL actor) to another, or by restructuring components. This perspective allows us to refine the goals identified in a GRL model into greater detail, as necessary. Generally, when creating these models we would iterate between both views. That is, we cannot decide on the allocation of goals to actors simply within, eg, a GRL model, but only after repeatedly refining both perspectives.

3.2 Feature Interaction Analysis

In our adoption of URN, we model features as goals, and service providers as actors/components. The methodology proposed in [9] includes three steps:

1. Model the features to be analyzed as a GRL goal graph. Goal graphs allow us to represent features, and to reason about conflicts between them.

2. Analyze the goal graph for conflicts. Conflicts point to possible feature interactions, in particular, if a conflict "breaks" expected functionality.
3. Resolve the interactions. During this step, UCM models allow us to explore the different alternatives suggested by the GRL models.

Examples of applying steps 1 and 2 will be provided in Section 5.

4 Classification of Feature Interactions

We propose to classify feature interactions among web services by their type, and their cause. We thus position our classification in the tradition of existing classifications of feature interactions for the telecommuncations domain [2], while emphasizing web service-specific aspects. A classification of web service feature interactions is beneficial as it allows their avoidance, detection, and resolution in a *generic* manner for each type of interaction. Solutions for specific feature interactions can then be generalized to other interactions of the same category.

Our classification is an extension of the work by [2] for the telecommunications domain. That work was based on the premise, even more important now with web services, that service creation is no longer governed by a single organization. It also discusses a categorization by *nature* of the interactions, which depended on the nature of the features involved ,the number of users, and the number of components in the network. However, some causes of feature interactions do not carry over to the web services domain. So we have dropped "Limitations on Network Support" as a cause, since explicit service invocation in web services possibly avoids all signaling ambiguity.

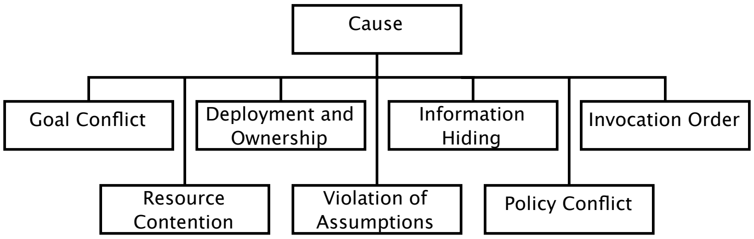

Fig. 3. Classification by cause

In our classification, a first distinction is made between functional and non-functional interactions with [9]. This distinction reflects that many of the side-effects are, in fact, of a non-functional nature, that is, they affect service properties such security, privacy, or availability. What this paper adds is a classification by cause shown in Fig. 3. It introduces two causes of interactions that we consider specific to web services, and are not encountered in closed, centralized telecommunications systems: deployment and ownership, and information hiding.

Goal conflicts take the shape of conflicting (non-functional) goals. They often occur as a result of unanticipated side effects, where in trying to achieve one goal, we inadvertently negatively impact another goal. We can indicate side effects in a GRL diagram by using a positive contribution link for the first, and a negative correlation link for the second goal. *Resource contention* is the fact that the use of some resource by a service makes it unavailable to another. It may result in service availability issues. *Deployment and ownership* decisions (where services are deployed, and who provides them) lead to performance, scalability and quality assurance issues, as well as conflicts of interest.

Assumption violations are caused by services that make incorrect assumptions about how another service works, and can, for example, be due to semantic ambiguity (use of the same concepts in different ways), or the presence of different versions of the same service. A result of *information hiding* is that service users cannot control how a service is implemented. This can lead to duplication of effort, inconsistencies, and even incorrect execution. *Policy conflicts* arise over contradictory policies that govern the behavior of services. Finally, *invocation order* is about features being invoked in a incorrect order, which may cause features to become ineffective, or timing glitches causing intermittent errors.

5 Selected Examples of Interactions

In this section, we provide examples of interactions to illustrate the classification. The context is a fictitious virtual bookstore, described in two parts:

1. We first describe the individual web services that will be used by the application. These services are developed without knowledge of how they will be composed later. Often they include third-party services that provide certain supplementary functionality such as identity management or payment.
2. We then create a composite service for an virtual bookstore from these services. We analyze the feature interactions that can occur as a result. As the services have been implemented independently they may embody assumptions that cause unexpected behavior as the services are composed.

This purpose of this division is to reproduce the problems that can result in the actual development of service-based applications. Each web service/feature is implemented with developers making assumptions that are individually valid, that is they faithfully implement the service interfaces. Feature interactions only result when these services are composed, often in unanticipated ways.

5.1 Examples of Features

The following features have one aspect in common: they all focus on one narrow type of service, and are usually employed in a supporting role. Examples of such supplementary services are identity management, payment processing, or shipping. In principle, any of these services could be provided by the requesting actor, but usually at a significant development cost, or risk of poor quality.

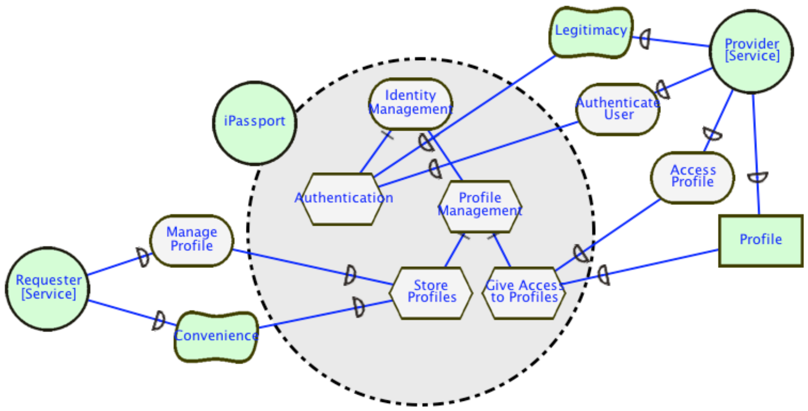

Fig. 4. GRL model of the iPassport feature

iPassport. The first example is an identity management feature. Identity management simplifies authentication with multiple service providers. It allows service requesters to authenticate themselves once with one service provider, and to access other service providers related to the initial service provider through a circle of trust. It simplifies the implementation of service providers, as well, because they no longer need to provide their own authentication component.

Fig. 4 is a GRL model of the iPassport feature. It models the service as well as each type of client as an actor (circle). As the diagram shows, iPassport mediates between Requesters [Service] and Providers [Service], and acts thus as a kind of broker. Requesters [Service] use iPassport to manage their profiles through the Manage Profile service, while Providers [Service] can authenticate users and access their profiles through the Authenticate and Access Profile services. Service provisioning relationships are modeled as functional dependencies. Functional requirements are represented as goals (rounded rectangle) that an actor wants to achieve. As shown, the dependencies are not restricted to interfaces. They also include non-functional and resource dependencies, for example, iPassport ensures

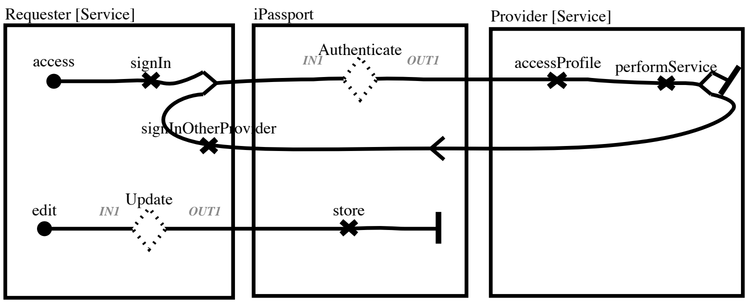

Fig. 5. UCM model of the iPassport feature

the requester's Legitimacy. Non-functional requirements are specified as softgoals (clouds) that cannot be achieved in an absolute manner. Resources (rectangles) represent physical or informational entities that must be available.

Internally, the iPassport feature is composed of an Authentication and a Profile Management features, which is responsible for storing profiles, and giving access to profile information. These features are shown as tasks (hexagons) that specify ways of achieving a goal. They are related to the Identity Management goal via decomposition links. More insight into behavioral and deployment aspects of a feature (how the tasks are performed) can be gained from a UCM model.

Fig. 5 shows the UCM model for iPassport. Note that this is only a top-level model with placeholders (also known as stubs) for submaps that define the Authenticate and Update behaviors (not shown). For example, the diagram captures (in the feedback loop with signInOtherProvider) that one Provider [Service] can link to another outside of the user's control. In the diagram crosses represent responsibilities, filled circles start points, and bars end points of paths.

PayMe. Payment processing allows payers to make secure payments online, and simplifies credit card processing for payees, while contributing to increased sales for them. As shown in Fig. 6, the payment processing feature PayMe provides two service interfaces: one to the Payer [Order] to Manage Accounts, and one to the Payee [Order] to receive payment for an order. The Process Payment service includes functionality to submit order details, as well as to cancel payments.

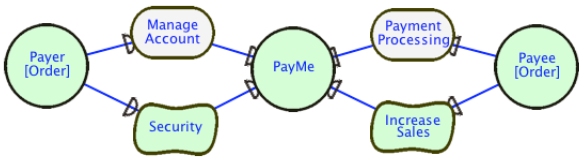

Fig. 6. GRL model of the PayMe service

Other Features. For reasons of space, we will sketch out the descriptions of the other features. The ShipEx feature provides a Delivery service to the Shipper [Order] with functionality for initiating shipment of an order, and canceling shipments, as well as a Tracking service for the Shippee [Order] to check on the status of a shipment. The EvilAds feature is an advertisement placement service, which provides a ClickAds service to any Host [Ad] that chooses to embed ads into its services. Finally, the Shark proxy service provides a Caching service through which a Provider [Service] can cache the results of popular service requests.

We implemented prototypes of these features, and tested them independently. However, space does not permit us to provide details of the implementation here, and we limit ourselves to describing the analysis of the observed interactions. Then we combined them into composite services, and analyzed the result for feature interactions. The largest of these case studies is described next.

5.2 Composite Service: Virtual Bookstore

The actor diagram for the composite service is shown in Fig. 7. The diagram models the Amazin virtual bookstore that gives Customers access to its virtual catalog, and the option to order books from the catalog through its Order Processing service. This service is composed from the features described above.

Amazin relies on a number of Suppliers to fulfill customer orders. Customer logins are handled through the iPassport identity management service, which provides an Authenticate User and an Access Profile service. On receiving a customer order, Amazin authenticates the customer, and accesses the customer's profile. It then selects a Supplier which stocks the ordered book and invokes its Order Processing service, passing along the customer's identity.

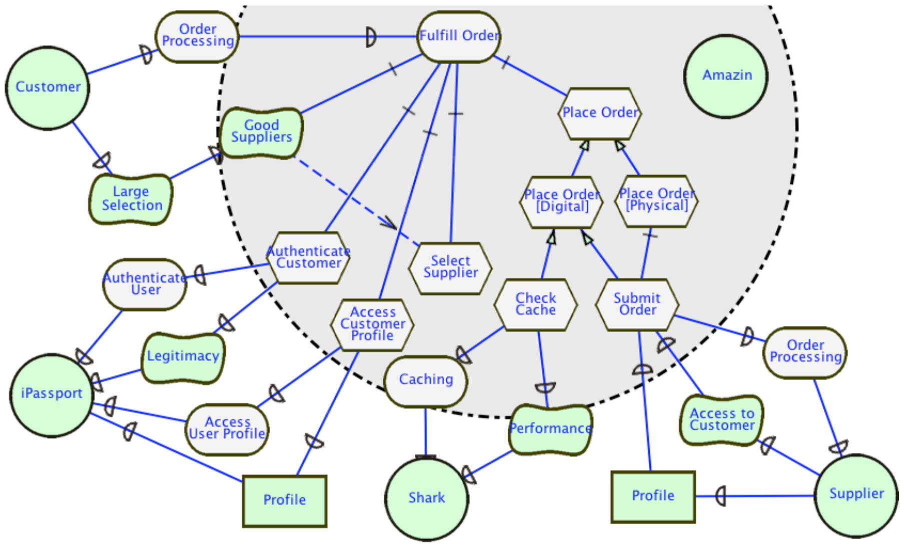

Fig. 7. GRL model of virtual bookstore Amazin

An internal structure of the Amazin service that fits this description is also shown in Fig. 7. This design makes assumptions that, while in agreement with service interfaces, may cause feature interactions. One potential source of interactions is the following optimization: in addition to physical books, Amazin also offers digital books for download, and it caches copies of popular orders.

The Supplier determines the availability of the ordered book, and, if successful, obtains the customer's payment and shipping preferences from the iPassport service. It then invokes the Payment Processing service provided by the PayMe financial service provider, and the Delivery service of Amazin's ShipEx fulfillment partner. Customers can track the progress of their orders via the Tracking service. They can also manage their online profiles, and accounts through services.

If a Supplier cannot fulfill an order, it will attempt to satisfy it from its network of Other Suppliers. Although omitted from the diagram, the chosen Other Supplier will use the same payment and delivery services as Supplier. Finally, some Suppliers choose to share selected customer information to an EvilAds advertisement agency via its ClickAds service as an additional source of revenue.

In the following we use this application to provide examples of the different causes of interactions identified in Fig. 3, as well as their types.

5.3 Goal Conflict

One conflict arises between Manage Profile and Access Profile, and is of type non-functional. As refined GRL model of the interaction is given in Fig. 8. The Convenience and Privacy goals of the Customer conflict with one another, since any iPassport member organization can access the profile, including those organizations with whom the customer has no trusting relationship.

While there is a trusting relationship between Customer and Amazin, the relationships between Customers and Suppliers are untrusted, and there is no guarantee that a Supplier will adhere to Amazin's privacy policy. Instead, it could decide, as an example, to sell the profile information to the target marketer EvilAds, which will then target the Customer with unsolicited ads.

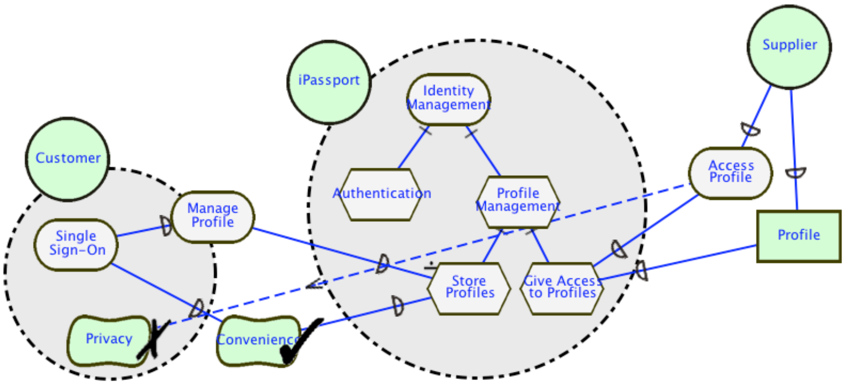

Fig. 8. Goal conflict between Manage Profile and Access Profile

This interaction is an example of a goal conflict, but it can also be classified as caused by deployment and ownership, information hiding, or a policy conflict. Deployment and ownership because at the root of the problem is one and the same entity (iPassport) authenticates the Customer and controls access to its profile. Profile information is shared between Amazin and its Suppliers without involvement of the Customer (as the UCM model in Fig. 5 clearly shows). Information hiding since the Manage Profile interface does not declare that profiles will be shared with parties the customer does not trust directly. Policy conflict because Suppliers are not bound to the same privacy policy as Amazin, whose policy is the only one the Customer has accepted explicitly.

5.4 Resource Contention

When Amazin invokes the Order Processing service of one of its Suppliers, this supplier will, in turn, place an order with one of its network of Other Suppliers, if it does not have the requested book in stock. However, this can lead to a situation where the order is sent back to Amazin itself, which is just an Other Supplier. Fig. 9 shows a scenario where Amazin is both a client, as well as a supplier to a given Supplier. If undetected, this can lead to an infinite loop of order requests, which could cause all actors linked via the loop to become unavailable. The dependencies at the source of the issue have been highlighted for emphasis.

This is a feature interaction between two implementations of the same feature, Order Processing, as implemented by multiple actors.

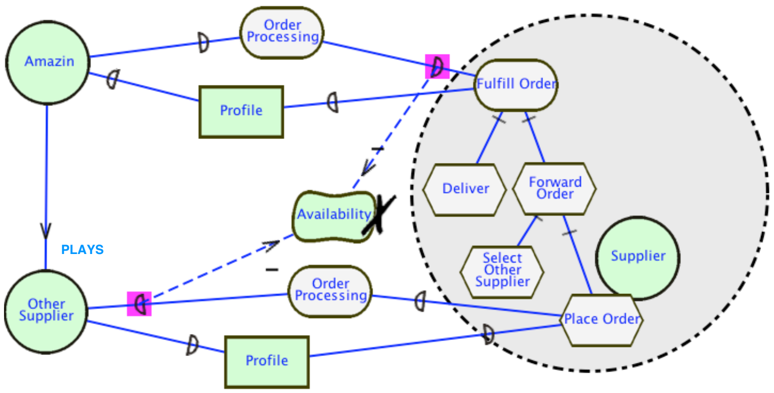

Fig. 9. Resource contention between OrderProcessing and OrderProcessing

5.5 Violation of Assumptions

The Caching service used by the Amazin service to keep local copies of digital content, and the Payment Processing service interact as result of a violation of assumptions. Caching digital content (in Shark, or another proxy) has the potential of preventing that access to the content will be properly billed. The Amazin service works correctly without caching, and thus an assumption may have been built in that for every order, a respective order will be placed with a supplier, and thus no internal accounting is required. If caching is added to improve the performance of the service, there is a potential that the implications of this change (breaking this assumption) are not fully understood by the designers.

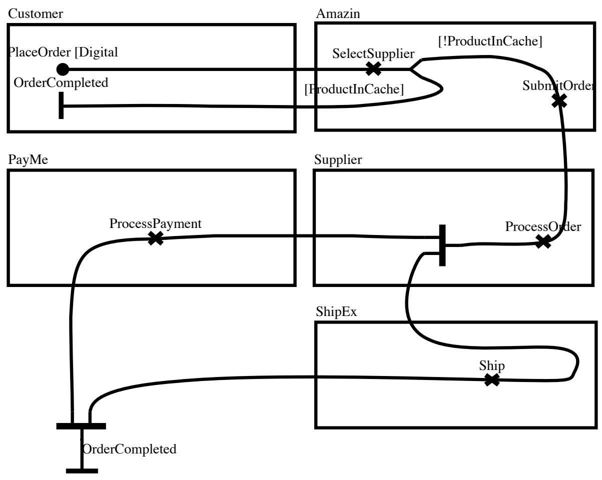

Fig. 10. Interaction between between Caching and Payment Processing–Delivery

The UCM model in Fig. 10 helps explain the situation. If ProductInCache is true, the return path in the upper left of the diagram will be taken.

5.6 Invocation Order

There is a potential conflict between Payment Processing and Order Processing, or Payment Processing and Delivery due to timing errors. The interaction can result in either the customer getting charged without the product shipped, or the customer getting the product for free. Both errors exploit timing glitches, for

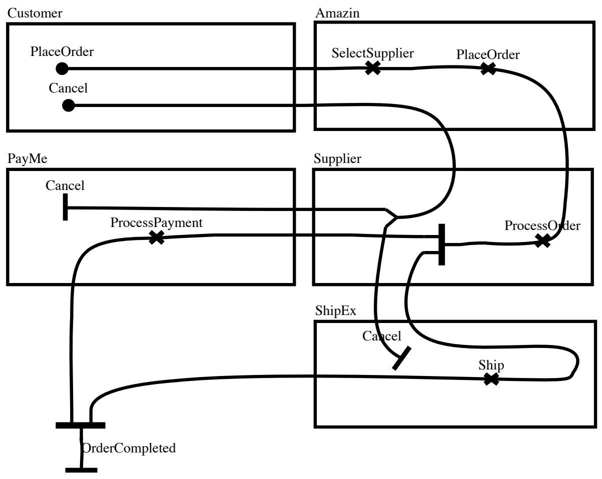

Fig. 11. Interaction between between Payment Processing and Order Processing

example, when the customer cancels their order, it could be that payment still gets processed (because Payment Processing was started before the order was canceled) but Delivery is aborted. The cancellation request was sent just before payment started, but arrived after Payment Processing has proceeded.

The UCM model in Fig. 11 provides the basis for understanding the cause of the interaction. The ProcessPayment and Ship responsibilities are initiated in parallel (the vertical bar after ProcessOrder indicates concurrency), and can execute in any order of one another. The Cancel requests to a component only take effect, if the ProcessPayment and Ship requests have not been started yet. This means that there are two successful cancellation scenarios, and two unsuccessful ones (where one of these requests has already been performed).

6 Related Work

In our earlier work [9] we have provided additional examples of interactions, as well as approaches for resolving them. By contrast this paper does not consider resolution. The goal conflict scenario is based on our work on assessing privacy technologies [10]. This paper also describes other privacy-caused interactions.

Liu und Yu [5] describe work on discovering privacy and security problems in P2P networks using GRL/*i**. Although they do not refer to feature interactions, they introduce a concept of conflict, and an extension to the GRL notation to indicate sources of conflict, as well as potential threats in a GRL model. In future work, we will integrate their results into our approach.

7 Conclusion

In this work we have presented work towards a classification of web service feature interactions. Our goal was to identify potential feature interactions in a composite web service. While some of these interactions can clearly be anticipated by service designers from past experience, it is impossible to plan for all circumstances during which interactions may occur, including interactions with services that do not even exist when a service is developed.

In our example, we therefore did not try to anticipate possible interactions, but focused on implementing the specified service interfaces. We can liken the development of web services to an iceberg. When we view a web service through its interface we only see the tip of the iceberg. While we can make the interface more specific, there will always be elements of its operation that escape the interface specification, just as the tip of the iceberg is no indication of its size.

More work on the classification is required. Our vision is that once a stable classification is in place we will be able to describe in a generic manner how interactions of services that fit a certain type and cause can be detected and resolved. Identifying and describing such patterns for detecting and resolving different kinds of interactions will set the agenda for our future work.

References

1. Amyot, D., Introduction to the User Requirements Notation: Learning by Example. Computer Networks, 42(3), 285–301, 2003.
2. Cameron, J., Griffeth, N., et al, A Feature Interaction Benchmark for IN and Beyond, Feature Interaction Workshop, 1–23, 1994.
3. GRL, http://www.cs.toronto.edu/km/GRL, last accessed in June 2005.
4. Keck, D, and Kuehn, P., The Feature and Service Interaction Problem in Telecommunications Systems, IEEE Trans. on Software Engineering, 779–796, 1998.
5. Liu, L., Yu, E., and Mylopoulos, J., Analyzing Security Requirements as Relationships among Strategic Actors, Symposium on Requirements Engineering for Information Security (SREIS), 2002.
6. O'Sullivan, J., Edmond, D., and ter Hofstede, A., What's in a Service? Towards Accurate Description of Non-Functional Service Properties, Distributed and Parallel Databases, 12, 117–133, Kluwer, 2002.
7. Ryman, A., Understanding Web Services, http://www.software.ibm.com/wsdd/techarticles/0307_ryman/ryman.html, 2003.
8. UCM, http://www.usecasemaps.org, last accessed in June 2005.
9. Weiss, M. and Esfandiari, B., On Feature Interactions among Web Services, International Conference on Web Services (ICWS), 88–95, IEEE, 2004.
10. Weiss, M., and Esfandiari, B., Modeling Method for Assessing Privacy Technologies, in: Yee, G., Privacy in e-Services, Idea Books, 2006 (to appear).
11. Weiss, M., and Esfandiari, B., On Feature Interactions among Web Services, International Journal on Web Services Research, 2(4), 21-45, October-December, 2005.