Pre-publication version.
To appear in *International Journal of Web Services Research*, 2(4), 2005.

On Feature Interactions among Web Services

Michael Weiss and Babak Esfandiari
Carleton University, Ottawa, Ontario, Canada
weiss@scs.carleton.ca, 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. However, the rapid introduction of new services paired with the dynamicity of the business environment also leads to undesirable interactions that negatively impact service quality and user satisfaction. In this paper, we propose an approach for modeling such undesirable interactions as feature interactions. As each functional feature is ultimately motivated by non-functional requirements, we make an explicit distinction between functional and non-functional features. We then describe our approach for detecting and resolving feature interactions among web services. The approach is based on goal-oriented analysis and scenario modeling. It allows us to reason about feature interactions in terms of goal conflicts and feature deployment. Three case studies illustrate the approach. The paper concludes with a discussion of our findings and an outlook on future research.

KEYWORDS:

Feature Interaction, Web Services, Goal-oriented Analysis, Non-Functional Features

1 INTRODUCTION

Web services promise to allow businesses to adapt rapidly to changes in the business environment, and the needs of different customers. However, the rapid introduction of new services paired with the dynamicity of the business environment also leads to undesirable interactions that negatively impact service quality and user satisfaction. In this paper, we propose an approach for modeling such undesirable interactions as *feature interactions*.

1.1 Feature Interaction Problem

The feature interaction problem has been first formally studied in the telecommunications domain. It concerns the coordination of features or services (we will not distinguish between features and services) such that they cooperate towards a desired result at the application level. The root causes for feature interactions in telephony systems are (Velthuijsen, 1993):

- Conflicting goals (services with the same preconditions but incompatible goals are in conflict, for example, services triggered by a busy extension)
- Competition for resources (services compete with each other for limited resources, which need to be partitioned among the services)
- Changing assumptions on services (services make implicit assumptions about their operation, which can become invalid when new services are added)

- Design evolution (services need to be added to meet new customer needs, and the system will need to interoperate with other vendors' systems).

A classical feature interaction is the interaction between Call Waiting and Call Forwarding on Busy. Both features trigger when the receiver of a call is busy, but only one of them should become active. This type of problem is usually resolved by introducing priorities. This most prominent implementation of this approach is the pipe-and-filter model (Utas, 2001), in which features are connected in a chain of filters in the order in which they get to process events.

The interaction between Outgoing Call Screening and Call Forwarding on No Answer is slightly more complex. Assume Alice is on Bob's outgoing call screening list (for instance, Alice could be the girlfriend of Bob's teenaged son Mark, and Bob does not want him to call her). But Mark quickly learns that he only needs to call his friend Joe, who temporarily forwards incoming calls to Alice. The solution to this type of problem involves confirming with the originating party, Bob, if Joe's forwarding the call to Alice is acceptable.

However, the feature interaction problem is not limited to the telecommunications domain. The phenomenon of undesirable interactions between components of a system can occur in any software system that is subject to changes. This is certainly the case for service-oriented architectures. First, we can observe that interaction is at the very basis of the web services concept. Web services *need* to interact, and useful web services will "emerge" from the interaction of many highly specialized services. Second, as the number of web services increases, their interactions will become more complex. Many of these interactions will be desirable, but other interactions may be unexpected and undesirable, and we need to prevent their consequences from occurring. As noted by (Ryman, 2003), many such interactions are related to security and privacy.

1.2 Web Services and Web Service Composition

Much research has focused on low-level concerns such as how to publish, discover, and invoke individual web services, as well as the security of web services. Other work has looked at dynamic web service composition, for example (Constantinescu et al., 2002), that is, at how higher-level services can be composed dynamically from lower-level services. Service composition raises a number of difficult challenges such as service description, selection, and orchestration.

At each of these stages (description, selection, and orchestration) we may experience undesirable interactions that prevent the proper performance of the service. However, there has been little research on managing such interactions at the level of the service logic. Most existing work is limited to managing the mechanics of the interaction (for example, enforcing a legal sequence of messages exchanged between the parties involved).

When composing web services, the functionalities provided by the component services must be considered. We also need to ensure that data and message types, sequence logic, etc. are compatible. However, as stated in (O'Sullivan et al., 2002), service composition amounts to much more than functional composition. Consideration must also be given to non-functional requirements such as privacy, and interoperability. For example when composing a personalized web service, we must also consider utility services such as identity management, and user profiling. But maintaining and sharing sensitive user information in a utility service raises privacy concerns.

A web service can be defined as a set of endpoints. An endpoint groups service operations, and each operation is defined by the messages exchanged to perform it. Web service composition languages such as the Web Services Flow Language (WSFL) define the notions of activities and workflows. Workflows define a partial order in which activities can be performed. Each activity can itself be implemented by a workflow in another organization.

The appropriate metaphor for thinking about composition of web services is therefore not the pipe-and-filter model of traditional telephony systems (Utas 2001), but that of a *flow system* with a richer behavior. Flow systems have three types of components: processing stages that can be connected in a variety of ways (not just sequences), data representations that are exchanged between stages, and orchestrators (engines) that coordinate the flows.

Up to this point we have only considered explicitly composed web services. These are best modeled as flow systems. However, as Figure 1 shows, an equally, if not more important category is that of implicitly composed services. While they are not intentionally composed, they can still interact in undesirable ways. We distinguish two subcategories: parallelism, and side-effect. In parallel composition, features are independently deployed, but may interact. This case is often encountered in the "traditional" analysis of feature interactions. Side-effect composition looks similar to flow composition, but the composed features are at different abstraction levels. A lower-level feature (perhaps from a third party) that implements a higher-layer feature may have unanticipated side-effects. The latter case is of particular relevance, as the case studies illustrate.

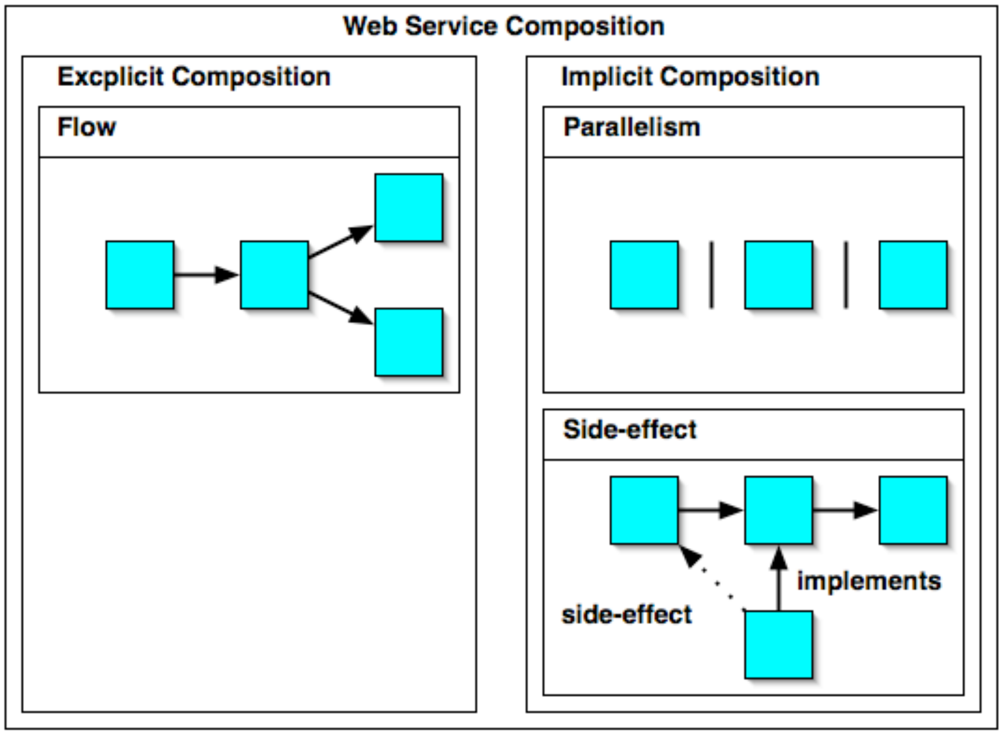

Figure 1. Explicit and implicit web service composition

Our focus will therefore be on feature interactions in the domain of web services. However, at the same time, we believe that our results will be applicable beyond this domain. Our reasoning is that web services are currently being deployed in a very rapid, decentralized, if not to say ad hoc manner. Problems due to conflicting goals, competition for resources, changing assumptions, and design evolution will therefore become visible much more quickly, and need to be resolved in a much shorter timeframe than for telephony features developed under central control. Lessons we learn from web services can then be applied back to the telephony and other domains.

1.3 Organization of the Paper

In this paper, we propose an approach for modeling undesirable interactions between web services as feature interactions. As each functional feature is ultimately motivated, at the business level, by non-functional requirements, we first make an explicit distinction between functional and non-functional feature interaction. We then describe our approach for detecting feature interactions among web services. It employs the User Requirements Notation, or URN (URN, 2003), to model features. This notation allows us to reason about feature interactions in terms of goal conflicts, and feature deployment. Three case studies and an e-commerce example illustrate the approach. The paper concludes with a discussion, and an outlook on future work.

2 FUNCTIONAL AND NON-FUNCTIONAL FEATURE INTERACTIONS

There is a growing recognition of the critical role of what are alternatively called business goals, qualities, or *non-functional requirements* (NFRs) in system development. Chung (1991) defines non-functional requirements as constraints over the functionality of a target system. This definition includes properties such as performance, security, or maintainability. Achieving non-functional requirements can be as crucial to system success as providing its functionality.

Chung et al. (2000) see the role of non-functional requirements as criteria for selecting between design alternatives that provide the same functionality. They model both functional and non-functional requirements as goals to be achieved by the design of a system. These goals are often in conflict with each other, and the objective of design is to find the right balance between them that satisfies *all* relevant goals (functional or non-functional).

With a specific focus on web services, O'Sullivan et al. (2002) consider non-functional "properties" of services an essential part of their description for the purposes of service discovery, negotiation, substitution, composition, and management. Their definition of non-functional properties includes billing and payment methods, provisioning channels, availability, service quality, security, trust, and rights.

The *Build Business Architecture First* pattern described in Arsanjani's (2002) pattern language for web services architecture motivates an approach in which business goals are mapped to services. The reason is that ultimately services must relate back to the business value created. Motivated by their work on Multidimensional Separation of Concerns, Hailpern and Tarr (2001) also differentiate between functional and management interfaces of web services. Management interfaces permit control over non-functional service properties such as performance, monitoring, and class of service that cross-cut all functional interfaces.

Thus, each functional feature is ultimately motivated, at the business level, by non-functional requirements. Similar to the distinction made in a recent workshop on feature interaction in composed systems (Pulvermüller et al., 2001), we make an explicit distinction between functional and non-functional features. The cross-cutting nature of non-functional features underlying this distinction is illustrated in Figure 2. Below, we define what we mean by functional, and non-functional feature interactions. We also provide examples of each type of interaction.

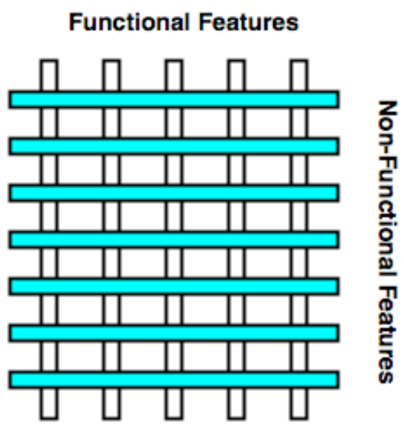

Figure 2. Interaction of functional and non-functional features

2.1 Functional Feature Interactions

Most interactions studied in the context of "traditional" features are of a functional nature. An example from telephony is an interaction between Call Waiting and Call Forwarding on Busy. A web services example is a race condition between an Order Completed and a Cancel message, which could result in situations like that where, due to timing delays, an order is shipped, but payment is cancelled. Functional feature interactions occur when functional features are composed. As identified in Section 1.2, this includes both explicit and implicit compositions.

Figure 5 in Section 4.1 shows a more detailed example of a functional composition of features. A new Personalization feature is constructed by composing Profiling, Information Filtering, and Identity Management features. Profiling takes care of managing user information. Information filtering is used to make query results more relevant to the user. Finally, identity management provides the user with a unique identity through which it can be identified to service providers.

Functional feature interactions can occur due to a number of reasons listed in Table 1. Generally, we found that these are not very different for web services than for other types of features. They include the "usual suspects": order of invocation, race conditions, overlapping guards, assumption violation (in particular, due to versioning, and semantic ambiguity), and resource contention. However, some reasons (such as assumption violation) seem to play a more prominent role in web services due to their dynamic, decentralized nature.

Reason	**Example**
Order of invocation	Order between a compression and an encryption service impacts the ability to compress.
Race conditions	Between simultaneously sent Order Completed and Cancellation messages.
Overlapping guards	Can occur between subflows of a composite web service that "fire" on the same event.
Assumption violation	Third-party services invoked by an intermediary with incorrect or incomplete arguments.
Resource contention	Due to service hosting, that is, how the service deployer manages class of service.

Table 1. Types of functional feature interactions between web services

2.2 Non-Functional Features and Feature Interactions

However, functional features are ultimately motivated by some non-functional, or system-level concerns such as privacy, security, or usability. For example, the Personalization feature of Figure 5 in Section 4.1 has the goal of enhancing the usability of an information service (from the perspective of the service user). Thus, if functional features are composed, the composition of these features will also impact the satisfaction of system concerns.

It therefore makes sense to talk of non-functional feature interactions, and even of *non functional features*. Non-functional features are system concerns affected by functional features, and, vice versa, they impose constraints on how the functionality is provided. With (Pulvermüller et al., 2001) we may consider functional features to be the functional *units* of a system, and non-functional features its non-functional *properties*. Asking whether a feature is an identifiable unit of a system, or a property can help us decide about the nature of the feature.

For example, we could be designing an instant messaging (IM) feature. Now we would like to secure the messages exchanged. We thus add encryption and decryption features that are invoked before sending, and after receiving an instant message. However, enabling these features has a significant negative impact on the performance of the system. Thus, there is a trade-off between performance and security. If we model performance and security as non-functional features, we can treat this trade-off as a feature interaction.

2.3 Where to Draw the Line?

The distinction between functional and non-functional features is not always clear-cut. Also, what is a non-functional feature to one party (for example, security), may be a functional feature for another party (if that party happens to be a security service provider). Features such as privacy, performance, or usability are clearly non-functional from an end-user perspective. Features such as order processing, or catalog aggregation are clearly functional.

However, there are features between both extremes. Features like billing, payment, or spellchecking have aspects of both functional and non-functional features. On the one hand, they are supportive, and cut across other features. On the other hand, they are not strictly properties of a system, but implementation units. Because these features impose constraints on functional features, they can be viewed as properties of the system, and thus as non-functional features, although they can still be implemented as a standalone service.

3 TOWARDS A DETECTION METHODOLOGY

Our approach employs the User Requirements Notation (URN, 2003) to model features. This notation allows us to reason about feature interaction in terms of goal conflicts, and feature deployment. Its focus is on user requirements (goals and functions), but it also enables their refinement into system requirements (Amyot, 2003). URN is comprised of two complementary notations: the *Goal-oriented Requirements Language* (GRL) (GRL, 2003), and the *Use Case Maps* (UCM) notation (URN, 2003). GRL is used to model business goals, non-functional requirements, design alternatives, and design rationale, whereas UCMs allow the description of functional requirements in the form of causal scenarios.

GRL build on the well-established goal-oriented analysis techniques introduced by Mylopoulos et al. (1999) and Chung et al. (2000). In goal-oriented analysis, both functional and non-functional requirements are modeled as goals to be achieved by the design of a system. During the analysis a set of initial goals describing the requirements is refined into a goal graph. This goal graph also shows the influence of goals on each other, and can be analyzed for goal conflicts. The objective of the design then becomes resolving these conflicts in a way that satisfies all initial goals.

Goals describe the objectives that a system should achieve. In GRL, these are also known as intentional elements. We call them intentional, because they allow us to answer questions such as why certain goals were included in the requirements, what design alternatives were considered, and why one alternative was chosen over another. There are four types of intentional elements in GRL: *softgoals*, *goals*, *tasks*, and *resources*. GRL also has support for modeling actors, which can have goals, and dependencies between actors. These elements will be used in Section 6.

Softgoals are used to represent non-functional requirements, their shape suggesting that there are no clear-cut criteria for determining when they have been achieved. Goals represent functional requirements. Tasks are solutions that achieve softgoals or goals. Resources are entities that need to be available to perform a task or achieve a goal. Figure 3 shows the symbols used by the notation. Further details of the notation will be explained as they are used in the case studies.

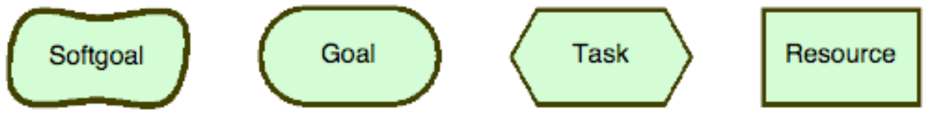

Figure 3. Notation for representing goals in GRL

One of the stated goals of URN is to describe scenarios without the need to commit to system components (Amyot, 2003). This capability is provided by UCMs. The basic notational elements for representing scenarios in a UCM are *responsibilities*, *paths*, and *components*, as shown in Figure 4. A *scenario* is a partially-ordered set of responsibilities that a system performs to transform inputs to outputs while satisfying certain pre- and postconditions (Amyot, 2003). Scenarios progress along paths from start to end points. The order of the responsibilities on a path indicates their causal relationship. Paths can fork to represent alternatives, and also join alternative path segments. Responsibilities can be allocated to components by placing them within the boundaries of that component. This is how we will be modeling feature deployment.

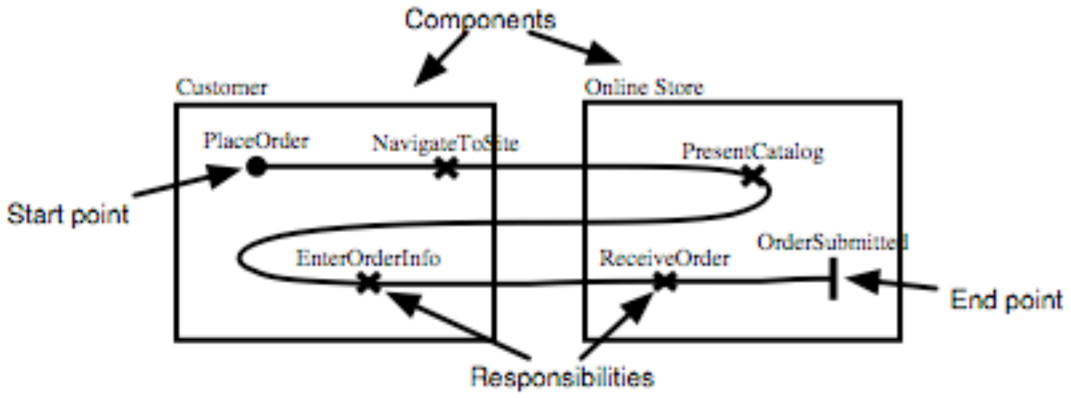

Figure 4. Subset of UCM notation used

With UCMs, different structures suggested by alternatives identified in a GRL model can be expressed and evaluated by moving responsibilities from one component to another, or by restructuring components (Amyot, 2003). The ease and flexibility with which this can be achieved helps designers and stakeholders to stay focused on addressing core design issues. UCMs require less detail and effort than other notations such as activity diagrams to achieve this.

We will now outline the steps of a methodology for detecting feature interactions between web services. It should be understood that, at this point, this is not a complete methodology. However, it provided us with a frame of reference for our empirical analysis during the case studies.

- Start by modeling the features you wish to analyze as a *goal graph*. Model functional features as goals, non-functional features as softgoals, and solutions that help achieve a goal or softgoal as tasks. Any part of a feature outside the scope of your current analysis should also be modeled as a task (you can make it the focus of another analysis later).
- Analyze the goal graph for *conflicts among goals*. These become visible as a set of conflicting softgoals (such as security vs. performance), but can be traced back to tasks, that is, to particular implementations of a goal or softgoal. We find that often a solution proposed to address one softgoal ends up negatively impacting another softgoal.
- Resolve the interaction, if possible, using one of a variety of *strategies* to be discussed further in Section 5. These include: refactoring the goal graph, changing the invocation order of services, changing their deployment, creating a standalone service, or ensuring that the initiator of a service request is consulted to resolve ambiguities (negotiation).

4 CASE STUDIES

4.1 Personalized Web Service

Personalization enhances the usability of a web service. For example, the user's shipping address could be stored in a profile, and filled in automatically whenever the user submits an order form, but does not provide a shipping address. Similarly, the results of a query to an information service can be made more relevant by filtering them against the interests specified in the user's profile.

Personalization is particularly useful in a mobile e-commerce setting, where users are accessing information, making purchases, or monitoring the progress of an auction from their mobile devices. Serving them only information relevant to their current context makes the information service more valuable. The context can contain information such as the user's identity, profile, and location, as well as information not specific to the user such as the current time.

We can design the Personalization feature as a composition of three features, as shown in Figure 5: Profiling, Information Filtering, and Identity Management. Profiling takes care of collecting user information and storing it in a profile. Information Filtering is used to select the query results deemed most relevant based on ther user’s profile. Finally, Identity Management provides users with a unique identity through which they can be identified to the information service provider.

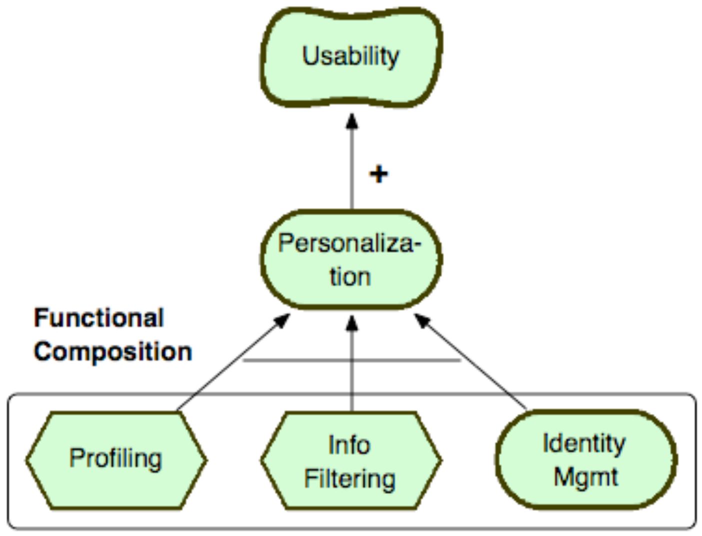

Figure 5. Components of a Personalization feature

Both Profiling and Information Filtering are represented as tasks in Figure 5. However, the Identify Management feature is modeled as a goal, since we are considering using a third-party implementation of this feature. That does not imply, however, that we could not find interactions involving the other two features. They are simply not the focus of the present analysis.

The Identity Management goal can be achieved or "implemented" in different ways. In goal-oriented analysis, these "implementations" are known as operationalizations. They constitute design alternatives, whose impact on system concerns we wish to analyze. In this case, our goal is to analyze the impact of our choice of a third-party identity management service in terms of (potential) undesirable feature interactions, and to devise remedies for resolving them.

Aside from the three functional features considered, there are many non-functional features involved when creating a personalized service:

- *Privacy*: Users need to disclose private information to the service provider, but they also want to be in control over who has access to which information.
- *Security*: Users expect their personal information to be protected from interception, and corruption on its way to and from the service provider.
- *Predictability (Trust)*: Users will trust a personalized service the more they perceive the query results as relevant and free of bias, and that their profile is not misused.
- *Usability*: While personalization can enhance the relevance of information, it can also put a burden on users in terms of how the user profiles are collected.

In line with the "standard" approach of treating the implementation of a third-party service as a *black box*, we select the Microsoft Passport service based on its documented service interface (Microsoft, 2003). Passport is one of several identity management standards, the Liberty Alliance standard (Liberty Alliance, 2003) being its main competitor. Passport authenticates service users to service providers, and gives providers access to the profiles of users.

We can model the integration of the Passport service as a task that satisfies the Identity Management goal as shown in Figure 6. Furthermore, we can decompose the Passport feature into two subfeatures, Authentication, and Authorization. Authentication identified the user to the service provider, and Authorization gives the service provider access to the user's profile. This is a very high-level decomposition, but sufficient for our analysis. The next step in our detection methodology described in Section 3 is to analyze the feature composition graph for conflicts.

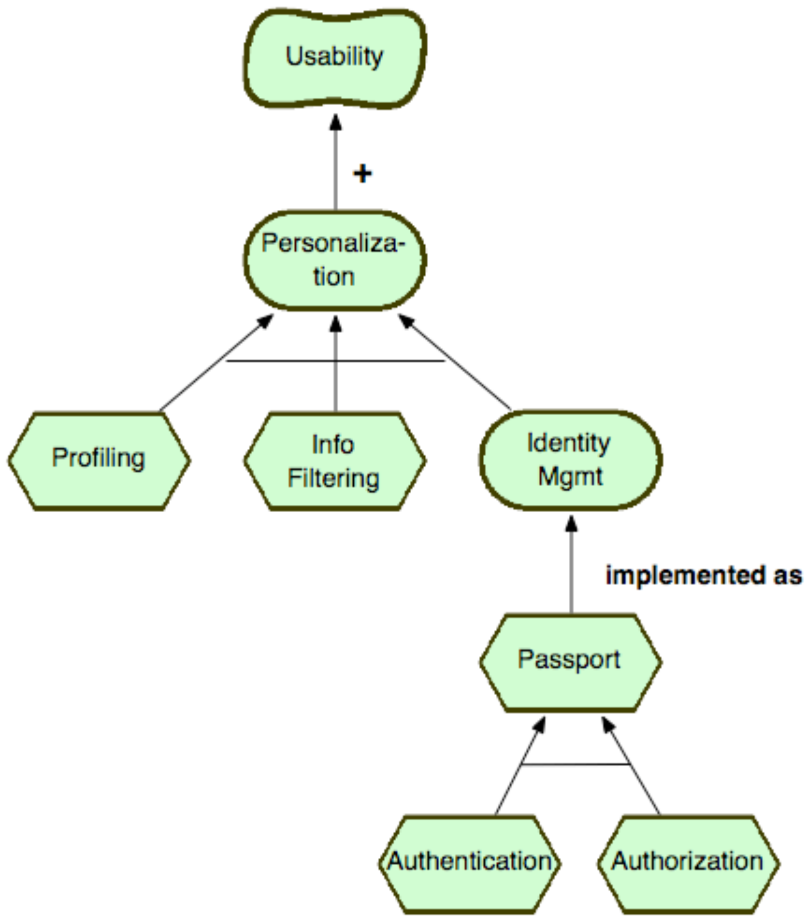

Figure 6. Implementation of the Identify Management goal

Conflicts often occur as a result of unanticipated *side-effects*. In goal-oriented analysis, these are shown as implicit contributions of goals (dashed lines in a goal graph). While explicit contributions are identified through decomposition (for example, Profiling to Personalization), implicit contributions (also known as correlations) show the impact of a goal on other goals than the one they refine. They are detected as the graph is developed. Another way of stating this is that the achievement of one non-functional feature (for example, usability) often affects other non-functional features (for example, privacy) in either positive or negative ways.

Returning to our example, the Authorization subfeature of the Passport feature gives service providers access to the user's information no matter who the service provider is. Specifically, in the current version of Passport (Snell et al., 2002) the user can only choose to mark sections of the profile as accessible by service providers, or inaccessible. No finer, provider-specific level of access control can be specified (for example, to only provide access to the age attribute to particular service providers, which the user trusts with this information). Furthermore, profile information can, effectively, be shared between service providers without the user's knowledge. While the user may have a trusting relationship with the initial service provider, it may not want its information shared with other service providers, for example, subcontractors of this provider.

As a result, the implementation of the Identify Management feature using Passport is found to violate ther user's privacy concerns. Figure 7 shows the negative side-effect of our particular implementation of the Personalization feature on Privacy as a correlation link. It is due to the Authorization feature, and consequently we show a contribution link from Authorization to Privacy. Upon further analysis, the reason can be seen in that Passport blurs the line between authentication and authorization. While it automatically authenticates the user to service providers, it also provides unconstrained access to the service user's profile.

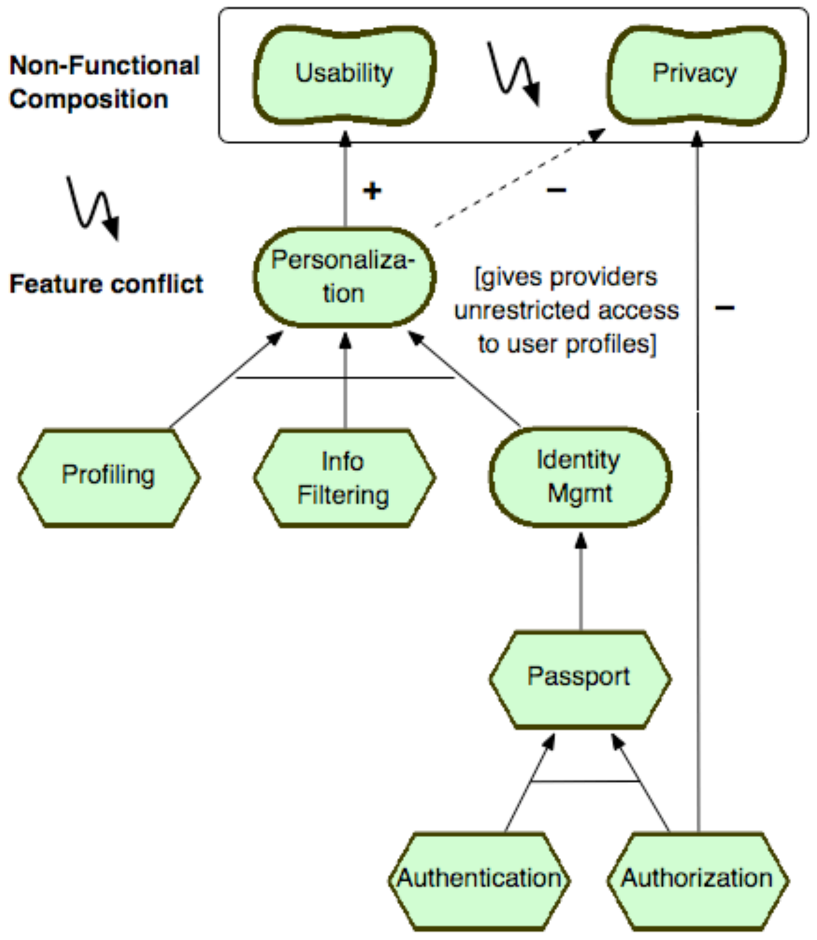

Figure 7. Conflict between usability and privacy in the Passport implementation

Our analysis does not stop here. Since goal-oriented analysis allows us to experiment with different design alternatives, its use is not limited to identifying a feature conflict. On the contrary, we can use it to suggest resolutions for the feature conflict. One strategy is to regroup the goals in the goal graph, perhaps adding new goals and tasks along the way. In object-oriented modeling such regrouping is also known as *refactoring*, and we will adopt this term here.

From our analysis we know that the feature interaction is caused by combining Authentication and Authorization in one feature, that is, under the control of one service provider. We can decouple those features by requiring that the implementation of Identity Management should *only* authenticate the user. An alternative design is shown in Figure 8. In My Identity Service, Authorization is implemented in accordance with the P3P (Platform for Privacy Preferences Project) standard (W3C, 2003). This design gives users control over what information they want to be shared with which service providers. To this end, we make Profiling a goal to be implemented using P3P. Its implementation of Authorization satisfies the user's Privacy goals.

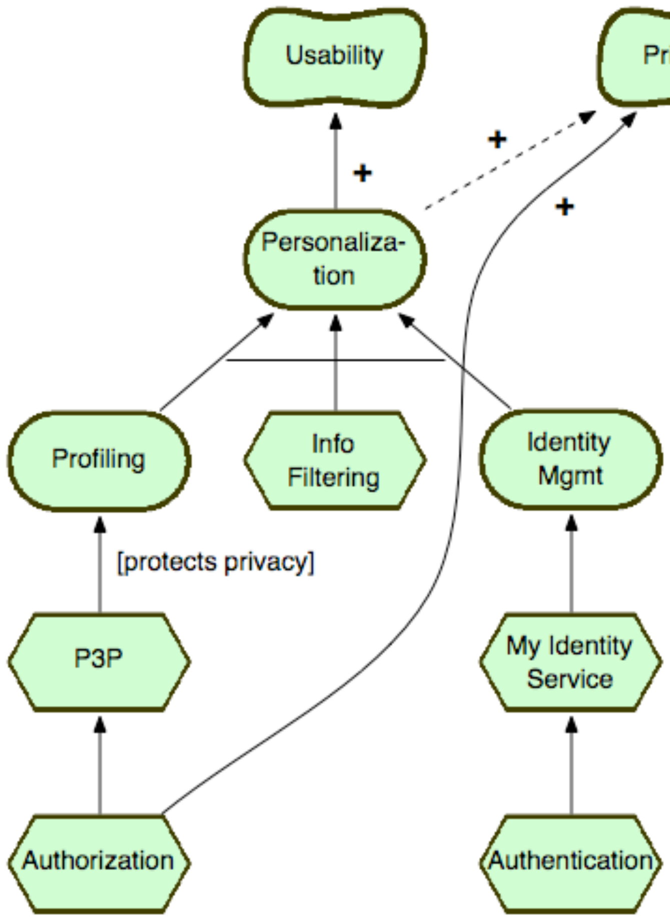

Figure 8. Resolving the Usability-Privacy conflict by refactoring the goal graph

The planned evolution of the Passport platform supports our analysis results. According to (Perkins, 2002; Snell et al., 2002), the upcoming version of Passport will include more measures for privacy protection. It is expected to integrate P3P to some degree. Users will be able to define policies for how their information should be shared, in line with our proposed feature interaction resolution. This example illustrates how our approach allows us to reason about feature interactions, and to explore refactorings of our design that resolve them.

Similarly, we could now look at the security issues associated with the Authentication feature, or any of the other non-functional features identified earlier for this service. It is important to note that what we had presented was one "view" of the system, and we intentionally limited the number of issues we wanted to deal with at once. While this is usually a good strategy, this is not always possible, for example, if we are dealing with interrelated non-functional features (for example, if security were somehow dependent on privacy). The next case study shows an example of where our analysis "forced" us to resolve multiple interrelated conflicts at the same time.

4.2 Restaurant Finder

Consider a business traveler looking for a restaurant to have dinner. She could consult a Restaurant Finder service to get suggestions that take her current location and preferences into account. The implementation of a Restaurant Finder service involves many aspects that make web services both powerful and difficult to implement. It is an example of a web service that must be aware of the user's context, will be dynamically selected, and may be part of a federation of web services in multiple locations. The latter two aspects were not covered by the Personalization service in Section 4.1. For these reasons it is often used as a reference example for the implementation of web services, for instance, by AgentCities (AgentCities, 2004), or Sun ONE (Sun, 2003).

At a high level, a Restaurant Finder service can be decomposed into two features as shown in Figure 9, Locate Service, and Recommend Restaurant. This decomposition hides details of how context-awareness and service transparency (transparent selection of the service) are achieved, as well as how the service might interact with Restaurant Finder services at other locations.

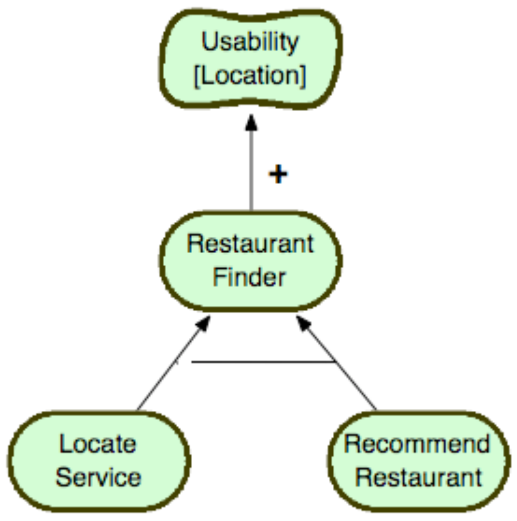

Figure 9. Components of a Restaurant Finder service

The Locate Service feature determines a local instance of the Restaurant Finder service at the user's location. It uses location information provided by the user's mobile device, and could be implemented by consulting a UDDI (Universal Description, Discover, and Integration) (UDDI, 2003) service registry. The Recommend Restaurant feature suggests a restaurant that matches the selection criteria (cuisine, price, rating, etc.) specified in the user's profile.

Where most implementations of a Restaurant Finder feature are going to differ is in how well they satisfy non-functional features. These include:

- *Usability (Location)*: Users want to be automatically directed to a Restaurant Finder service for their location (without having to enter their current location).
- *Usability (Service)*: Users expect that the most appropriate service is selected, if several Restaurant Finder services are provided in a given location.
- *Usability (Interface)*: Users do not want to deal with multiple service interfaces for different locations, but access the service from a common interface.

- *Predictability (Trust)*: Users will trust restaurant recommendations the more they perceive them as relevant, and free of bias, and that their profile is not misused.
- *Predictability (Quality)*: Users expect the results to be correct (for example, the distance of a restaurant from their current location), and personalized to their preferences.

The big selling point of a Restaurant Finder service is its location transparency. Since location transparency contributes to Usability, we model it as an aspect of Usability. In Figure 9, we express the concept of "usability due to location transparency" using the GRL concept of a subtype, Usability [Location] (read as: Location restricts the type Usability).

In a typical implementation of the Restaurant Finder service, Locate Service is implemented as a Directory that can be queried for service providers by name and type of service, as shown in Figure 10. Recommend Restaurant is implemented as a manually compiled listing of restaurants and ratings assigned by restaurant critics, the Listing and Ratings feature. This listing can be searched based on user-defined criteria (such as cuisine, price, rating, etc.).

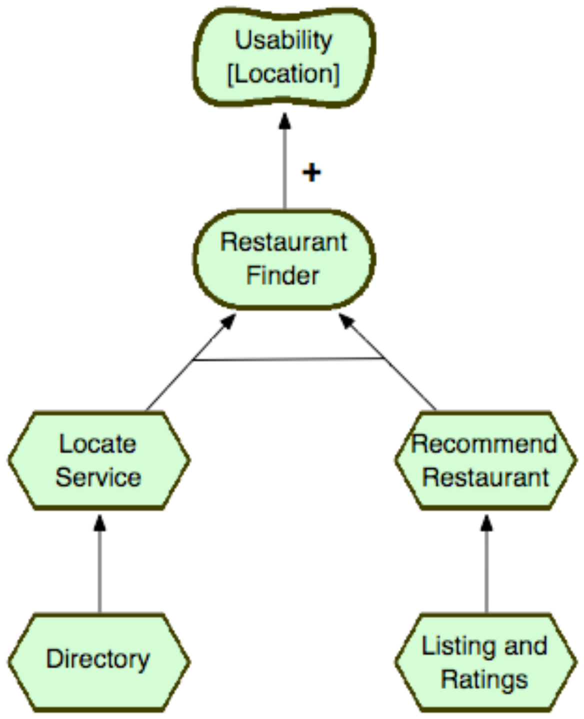

Figure 10. Implementation of the Locate Service and Recommend Restaurant goals

During the analysis of the feature composition graph, three side-effects are identified, as shown in Figure 11. The Directory feature does not protect the service user from the idiosyncrasies of the interfaces of local Restaurant Finder services. No common interface is provided to ensure access transparency, resulting in a violation of the Usability [Interface] feature.

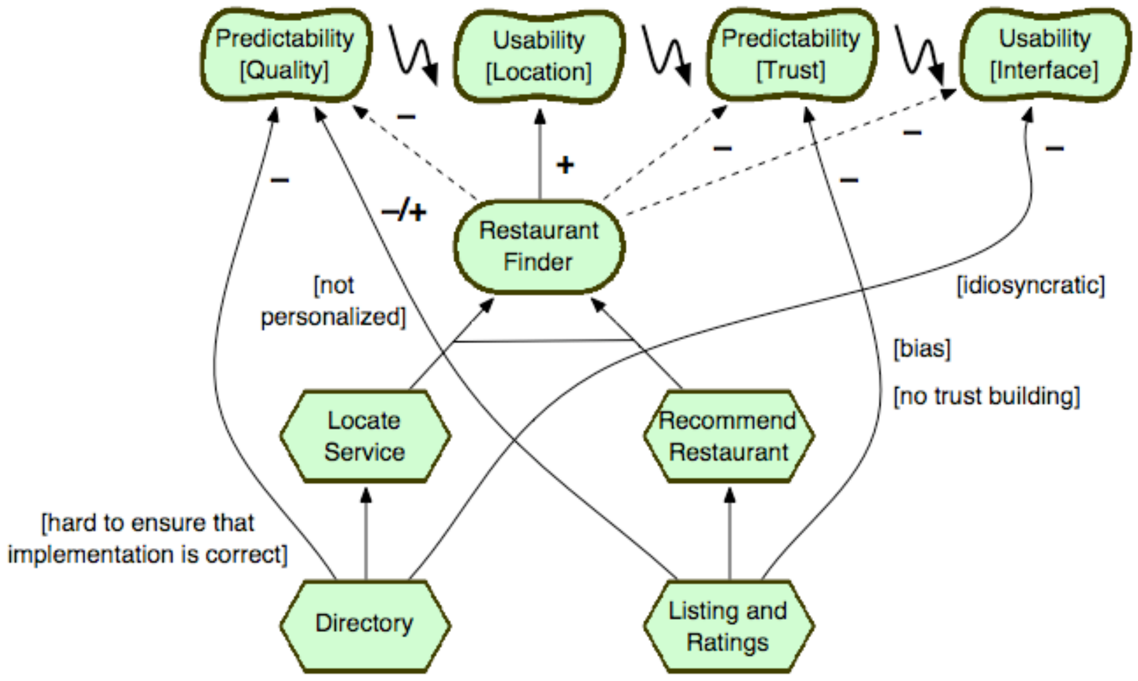

Figure 11. Conflicts between Location and Interface aspects of Usability, and Quality and Trust aspects of Predictability in the Restaurant Finder feature

It is also hard to ensure that service providers correctly implement the service interface. Directory only provides an index for searching service providers, but does not police service quality. That is, it can only guarantee that a service complies at a syntactic level with its interface. However, if the feature miscalculates the distance between user and restaurant, it could not be detected. This restriction is in conflict with the Predictability [Quality] feature.

The Listing and Ratings lacks personalization. This makes the service inconvenient to use (for example, it requires re-keying of search criteria), and hurts Predictability [Quality]. While it would relatively easy to extend the feature to remember the user's preferences, and use them in future searches, the main drawback of such personalization is that it reduces serendipity by pigeonholing the user. For example, the service will never suggest restaurants that serve a cuisine typical for the region, if it is not included in the user's preferences.

Limited personalization is also not a good basis for building up a trusting relationship with the user. Trust is something that can only be built over time by using a service. For example, a service for restaurant recommendations becomes trustworthy, after it has been recommending restaurants that the user has liked in the past (these are not redundant, but on the contrary, the basis for trust). In addition, the implementation suffers from the issue of bias: for example, there is no way to detect if the service only returns restaurants that have paid a fee for being included in the restaurant listing. Thus the feature is also in conflict with the Trust aspect of Predictability.

As in the previous case study, we can resolve most of the detected feature interactions by refactoring the goal graph. At the center of the solution is a trusted portal through which the user will interact with the service. This portal makes the selection of localized services transparent, and polices the quality of the recommendations. We are also not restricted to a centralized portal, but can use a federation of portals that share information with each other instead (however, this is not shown in the present graph). The resulting goal graph is shown in Figure 12.

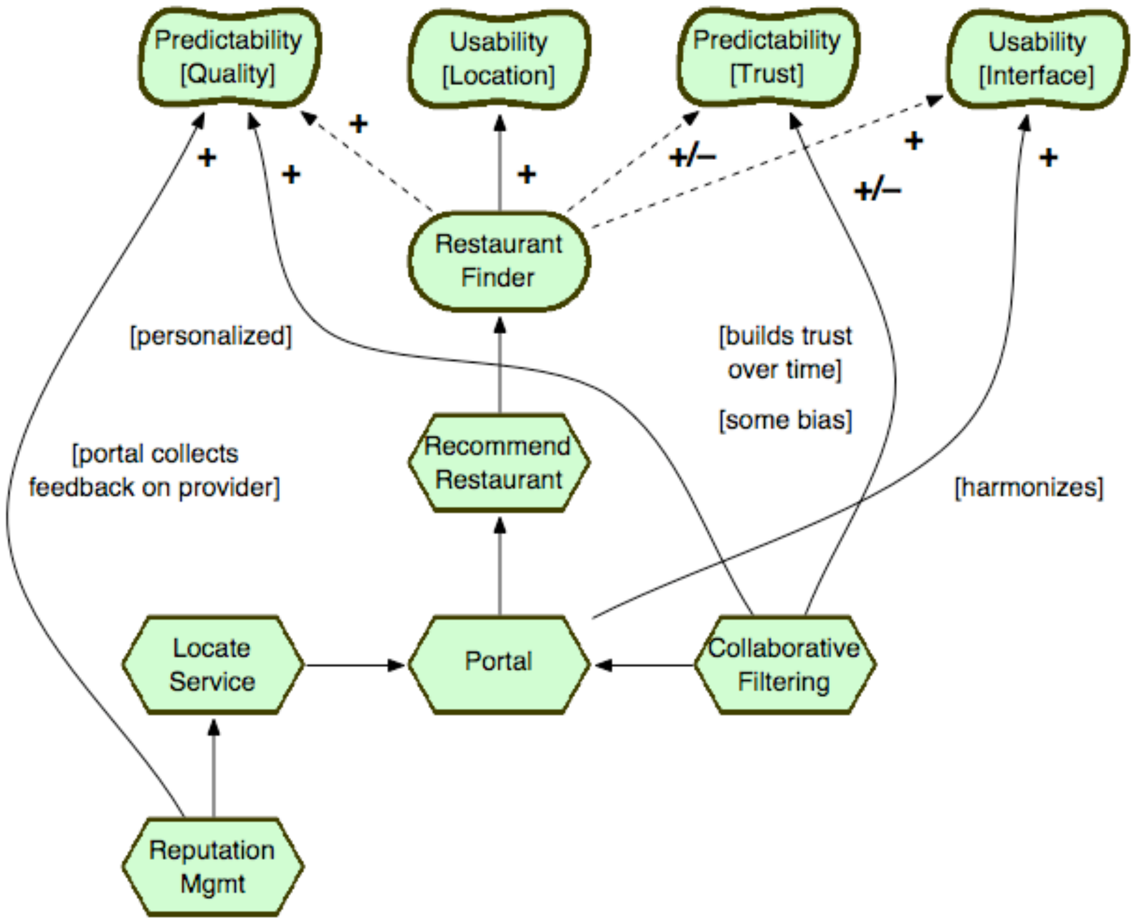

Figure 12. Resolving the conflict between Usability and Predictability

In this solution, users no longer need to interact with the Locate Service feature directly. The problem with the varying quality of service providers is also addressed by implementing Locate Service through a Reputation Management feature. This feature allows users to rate providers of the Restaurant Finder service. Based on this feedback a ranking of the service providers can be established, using a collaborative algorithm such as Sporas (Zacharia et al., 1999). The portal automatically selects the highest-ranked service for the user's location.

The problem of limited personalization is addressed by using a collaborative filtering mechanism to generate recommendations from a population of users. Recommendations are no longer just based on the user's query, or their profile information, but on the likes and dislikes of similar users. This type of recommendation encourages serendipity. The improved quality of the recommendations also builds up the user's trust over time. Finally, the portal puts a single, harmonized interface on top of the different interfaces of individual Restaurant Finder services. It provides a mapping layer for adapting the idiosyncrasies of a given service interface to a common interface.

A solution similar to the one proposed as result of our analysis has been proposed in one of the AgentCities projects. A recommender system based on opinion-based filtering method (Montaner et al., 2002) uses a collection of service and personal agents. Service agents offer information about restaurants, and personal agents provide users with recommendations on restaurants based on their interaction with similar *trusted* personal agents. The underlying trust model enhances the reliability of the recommendations. In this model, personal agents weigh the recommendations from trusted contacts higher than those of others.

4.3 Third-Party Intermediary

An *intermediary* is a component that sits between service users and service providers. It is itself a web service that provides a certain value-added (such as authentication, auditing, caching, or brokering). It works by intercepting requests from service users, performing its functionality (for example, authenticating the user), and forwarding the request to the server. Later, it intercepts the service provider's response, and relates it back to the service user. This corresponds to Web Services Architecture Usage Scenario S030 (W3C, 2002).

Consider the example of a Word Processing web service that makes use of two third party web services, Spellchecking and Formatting as shown in Figure 13. Suppose that in requests originating from the Word Processing feature the language option of the Spellchecker service is set to UK English. Formatting also happens to use a third party spellchecker service itself (which may be the same service). It assumes that American English should be used for spellchecking. However, following good information hiding practice, the Word Processing service is unaware of this.

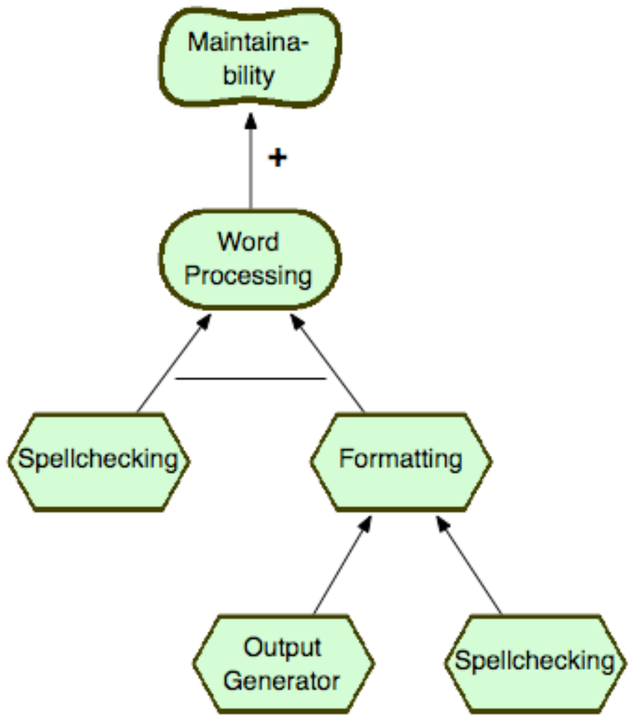

Figure 13. Components of a Word Processing service

The main value-added by this design of the Word Processing service is Maintainability. A consequence of information hiding, maintainability suggests delegating supplementary functions such as spellchecking and formatting to third-party services. This is a fairly realistic scenario given the existence of services like Google's spellchecking service (Google 2003). The same reasoning underlies the use of Spellchecking insider the Formatting service.

The intermediary in our scenario is the Formatting service. It executes Spellchecking on behalf of Word Processing, but does not disclose that. As Figure 14 shows, this leads to two types of conflicts. As the spellchecker gets invoked two times, this negatively impacts performance. More disastrous, though, for the correct performance of the Word Processing service is that Formatting overrides the user's language preference. Formatting assumes that an American English dictionary should be used, whereas the user preferred the UK spelling.

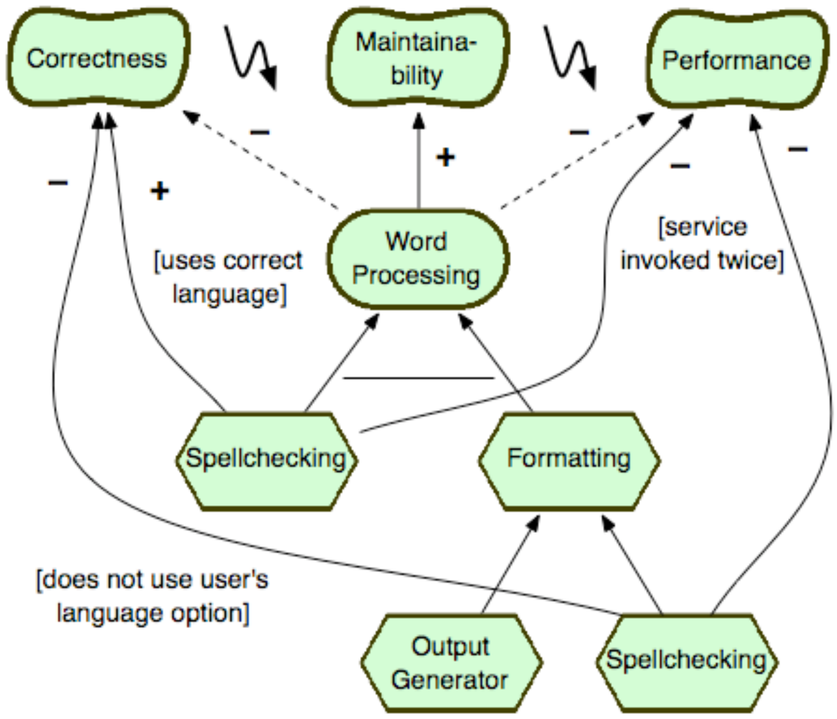

Figure 14. Conflicts between Maintainability, Performance, and Correctness in the Word Processing service

The problem highlighted by this case study is how information hiding by a web service implementation can lead to negative consequences for Correctness, and, in this case, Performance. The strategy for resolving the conflict calls for "breaking" the information hiding principle. The conflict can no longer occur, if each recipient of a service request (that is, Spellchecking) were to consult with the initiator of the request (that is, Word Processing) as to how to perform the service (in this case, which language option to use).

5 DISCUSSION OF THE CASE STUDIES

In this section we try to generalize our results obtained in the case studies. Our guiding questions for evaluating each case study are: what type of feature interaction, and what resolution strategies does the case study illustrate? We identified three types of interactions: goal conflicts, deployment and ownership issues, and issues related to information hiding. Goal conflicts are illustrated by the case studies in Sections 4.1 and 4.2. The impact of deployment and ownership issues is illustrated by all three case studies. Finally, information hiding issues are touched on by the case studies in Sections 4.1 and 4.3. Below we discuss these types of interactions.

5.1 Goal Conflicts

We found goal-oriented analysis to be particularly suitable for the analysis of feature interactions, not just in web services. This form of analysis hinges on a model of the functional and non-functional goals of a system. This model is provided by a goal graph, which captures the influence of goals on each other. Goal-oriented analysis allows us to *reason* about feature interactions as goal conflicts. (As features are modeled as goals, we have also referred to a goal graph as a feature composition graph on several occasions.)

Goal conflicts become visible as a set of conflicting softgoals. They often occur as a result of unanticipated side effects, where in trying to achieve one softgoal, we inadvertently negatively impact another softgoal. In our analysis, we trace conflicts back to the tasks that cause them. Such interactions can often be solved by refactoring the goal graph. In refactoring we regroup the goals, possibly introducing new goals and tasks to the graph along the way.

In the first case study, the interaction between Usability and Privacy is resolved by grouping the Authorization feature with the Profiling goal. This amounts to decoupling the Authentication and Authorization features. Although not shown in the goal graph, this amounts to changing ownership of the Authorization feature from Passport to the service user. A deeper justification for refactoring the goal graph in the way described is thus provided by an analysis of deployment and ownership issues (Section 5.2).

The second case study resolves a conflict between Usability and Predictability. The refactored solution replaces the original Listing and Ratings feature by a trusted portal, which in turn uses collaborative filtering to rate the restaurants. This improves the Predictability of the service. Again, it possible to justify this resolution in terms of deployment and ownership (Section 5.2). The Listing and Ratings feature is an indication of a conflict of interest, which can be resolved by decoupling the feature into two independently owned features.

In some cases we found that we need to supplement goal-oriented analysis with deployment models, and use case maps. However, as such, this finding is not surprising, and the modeling community has already recognized the need to create a notation that combines goal-oriented analysis (static aspects) with use case maps (dynamic aspects) (URN, 2003). The following sections illustrate how these other models can be used in conjunction with goal graphs.

5.2 Deployment and Ownership

The intrinsically open and distributed nature of web services implies that some decisions need to be made as to *what* services are needed, *who* should provide the services (ownership), and *where* services should be deployed (deployment). Deployment and ownership decisions influence feature interaction issues in the following ways:

- Physical decoupling of services (by making dependent services run on different hosts, and sometimes under different ownership) can help solve *resource contention* issues (by avoiding that a single host becomes a bottleneck).

- Grouping of related services gives their owner more control over the resulting system, and allows for performance optimizations. However, physical decoupling under the same ownership can lead to better scalability.
- Conversely, delegating a functional feature to a third party removes the need for local management features to assure its quality. Of course, we then have to trust the third party that such features are properly supported.
- Ownership of some services by the same owner can also lead to a *conflict of interest*, and a loss of trust in the owner due to (the perception of) bias. This provides further incentives for delegation and physical decoupling.

In the case study in Section 4.1, the reason for the feature interaction is that Authentication and Authorization are owned by the same service provider (Passport). The UCM in Figure 15 shows how the features are assigned to the parties involved before, and after redeployment. It should be noted, however, that, as a consequence of decoupling, the Personalization feature can no longer benefit from the profile storage feature that is implicitly offered by Passport.

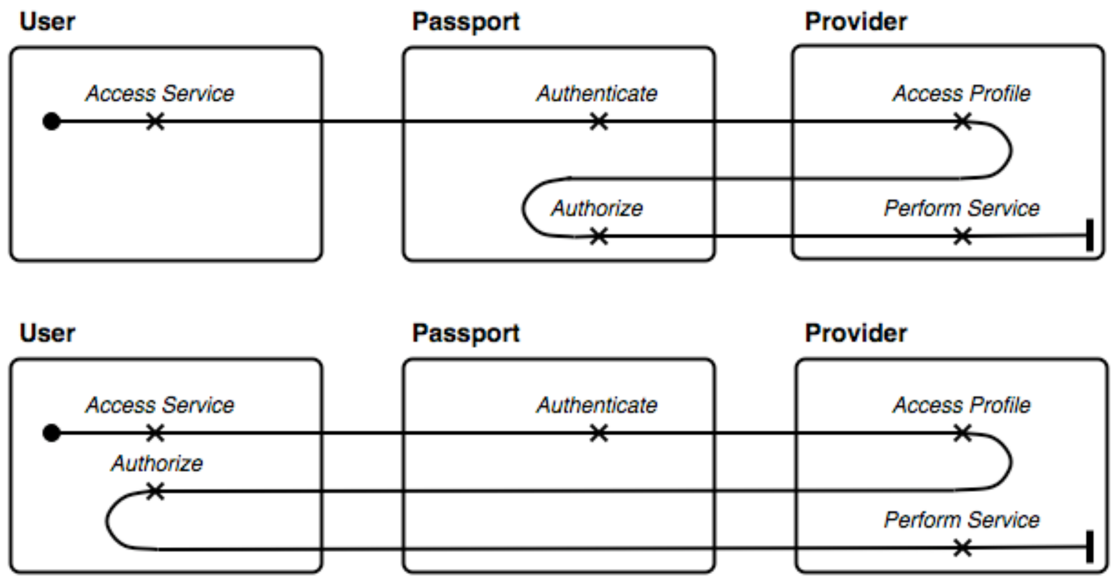

Figure 15. Before and after redeployment of Authentication and Authorization

In the case study in Section 4.2, the Restaurant Finder service provider should not provide both listings and ratings, as this would constitute a conflict of interest. Our resolution strategy suggests to decouple these tasks into two independently owned features. In our solution, both listing and rating services are based on user feedback. In this set-up, a biased service will receive a lower ranking over time than an unbiased one. The case study in Section 4.3 is an example in which delegating the spellchecking functionality to a standalone third-party service relieves the Word Processing service from having to assure that its own spellchecker is kept up-to-date. However, due to information hiding (Section 5.3), feature redeployment creates an unforeseen conflict.

5.3 Information Hiding

Information hiding is a software engineering practice which aims at hiding the complexity of a piece of functionality behind an *interface*. Since the functionality of a service can only be accessed through this interface, using the service does not require detailed knowledge of its implementation (ease of use), and the decoupling of interface and implementation promotes reuse. However, as a consequence of information hiding, users of the service cannot control how the service is implemented, which leaves less room for application-specific optimizations.

The lack of control over the implementation of a service also means that in a given composition, the same web service can end up being used more than once. In the worst case scenario, as shown by the Word Processing service, the hidden use of a service can override its explicit use. In the example, Spellchecking is also used by Formatting, but *after* the explicit invocation within Word Processing. If the hidden call uses incorrect or incomplete arguments, the service override means that the effects of the two invocations are in conflict with each other. Here, the implementers of the Formatting service had made the hidden assumption that users speak American English.

A similar example could have been given for a third-party service with a privacy policy that conflicts with that of the intermediary. In that scenario, the user trusts the intermediary, but is unaware of the privacy violation caused by information hiding. There are a number of ways of resolving such interactions, and we will discuss three of them. The first strategy is to assume a centralized web service management platform with complete access to all parties. A simple traversal of the service composition graph can then detect the conflict.

Given that in practice we do not have access to this information, consider an alternative strategy where the intermediary aggregates the input parameters to be supplied to third parties. In the Word Processing example, the Formatting service would report back the language option to be used by Spellchecking to the Word Processing service. In the privacy scenario, the intermediary could combine the privacy policies of the third parties, and present them to the service user who can then determine which information to release, or decide to provide different information to each service provider in separate parcels (W3C, 2003).

A third strategy is to require the final recipients of a service request (third parties) to consult with the initiator of the request (service user) as to how to perform the service. In the Word Processing scenario, the Spellchecking and the Word Processing services would agree on the language to use. It is then also possible for the service user to detect duplicate invocations of the same service in the same context, and to deny it. Similar reasoning can be applied to the privacy scenario illustrated by the Personalization service. The solution in (W3C, 2003) is for the service user to check the privacy policy of each third party for compliance with its privacy preferences.

The use case map in Figure 16 shows that the third strategy involves “breaking” the information hiding principle. Before invoking the Spellchecking service provided by the Spellchecker, the Formatting sercice (provided of the Formatter) discloses its intention to do so to the Word Processing service, which can then pass on the appropriate language setting (UK English).

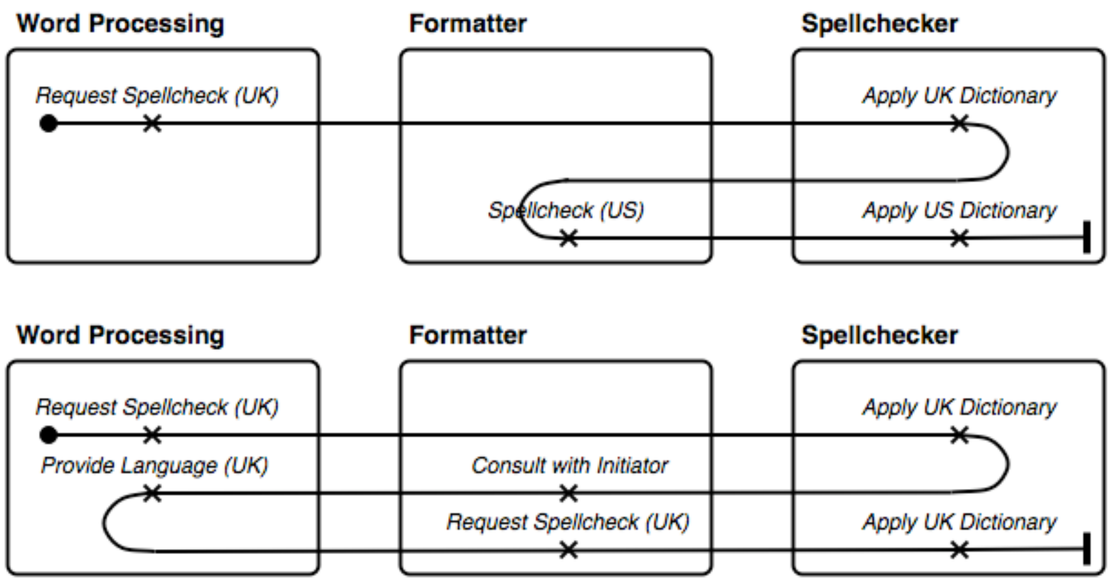

Figure 16. Before and after "breaking" the information hiding principle

5.4 Summary of causes of Feature Interactions in Web Services

Figure 17 summarizes the typical *causes* of feature interaction that are either specific to the web services domain (as discused in the sections above), or have been identified in previous research on feature interactions in the telecommunications domain (Utas, 2001), and also apply here. Our classification of feature interactions extends previous classifications in telecommunications by (1) distinguishing between functional and non-functional interactions, and (2) introducing two causes of interactions that we consider specific to web services: deployment and ownership, and information hiding. Neither of these are issues in closed, centralized telecommunications systems.

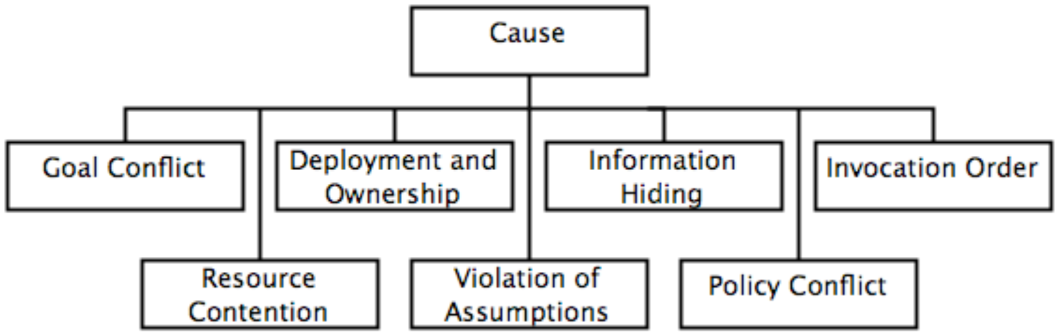

Figure 17. Causes of Feature Interactions in Web Services

6 APPLICATION TO E-COMMERCE

In this section we describe how the causes of feature interaction identified in Section 5.4 can be observed in a typical e-commerce application using web services. Figure 18 is a so-called actor diagram for a typical e-commerce application. An actor diagram shows the actors and their goal dependencies. The diagram models the Amazin online bookstore that gives Customers access to its virtual catalog, and the option to order books from the catalog through its Order Processing service. In the actor diagram, the Order Processing service is modeled as goal dependency, which states that the Customer depends on Amazin in order to achieve the Order Processing goal.

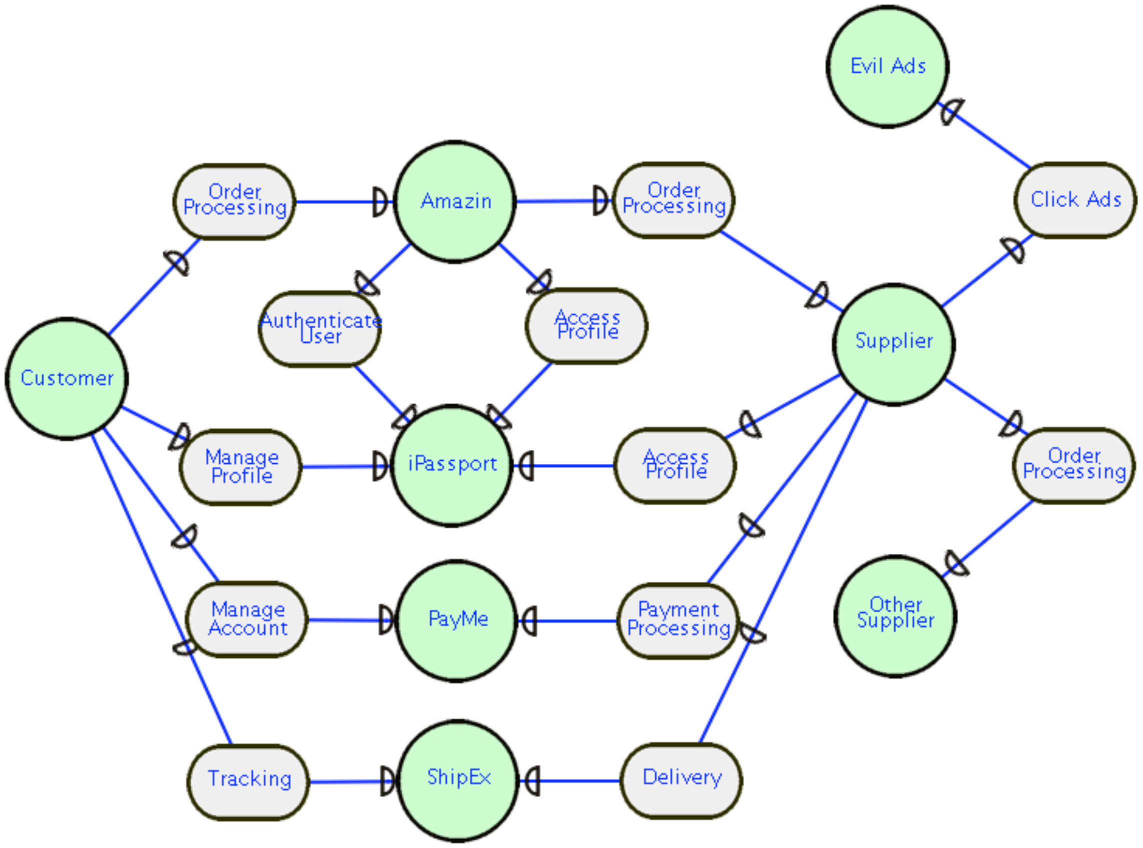

Figure 18. A typical e-commerce application: the **Amazin** online bookstore

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, then accesses the customer's profile. It subsequently selects a Supplier which stocks the ordered book and invokes its Order Processing service, in turn, passing along the customer's identity.

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 provided by ShipEx. They can also manage their online profiles, and payment accounts through services provided by iPassport and PayMe, respectively.

If a Supplier cannot fulfill an order, it will attempt to satisfy it from its network of Other Suppliers. Although not shown in the diagram, the selected Other Supplier behaves the same as a Supplier, that is, it will use the same payment and delivery services. (Properly modeled, Other Supplier would be a role that a Supplier can play. However, this is not supported in the current version of GRL.) Finally, some Suppliers might choose to share selected customer information with the EvilAds advertisement agency via its ClickAds service as an additional source of revenue.

To keep the application manageable we have avoided the kind of duplication and extraneous information in the models that one would observe in a real-world example. For instance, we chose not to model that Amazin might have its own inventory from which to fulfill popular orders, as this would not add anything new to model with regard to the feature interactions we wish to demonstrate. In the following, we provide examples of the different causes of feature interactions that can occur between the services modeled in Figure 18.

6.1 Goal Conflict

Amazin and its Suppliers obtain the customer's payment and shipping preferences from the iPassport service. While this is convenient for both Customers and service providers, there is also potential for undesirable side-effects. The goal graph in Figure 19 allows us to analyze the situation. It shows that the Usability and Privacy goals of the Customer conflict with one another, since any service providers registered with iPassport can access the profile, including those providers 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 EvilAds, which will then target the Customer with unsolicited ads. This is a non-functional feature interaction between the Manage Profile and Access Profile services. It is primarily an example of a goal conflict. However, it can also be classified as a deployment and ownership, information hiding, or policy conflict.

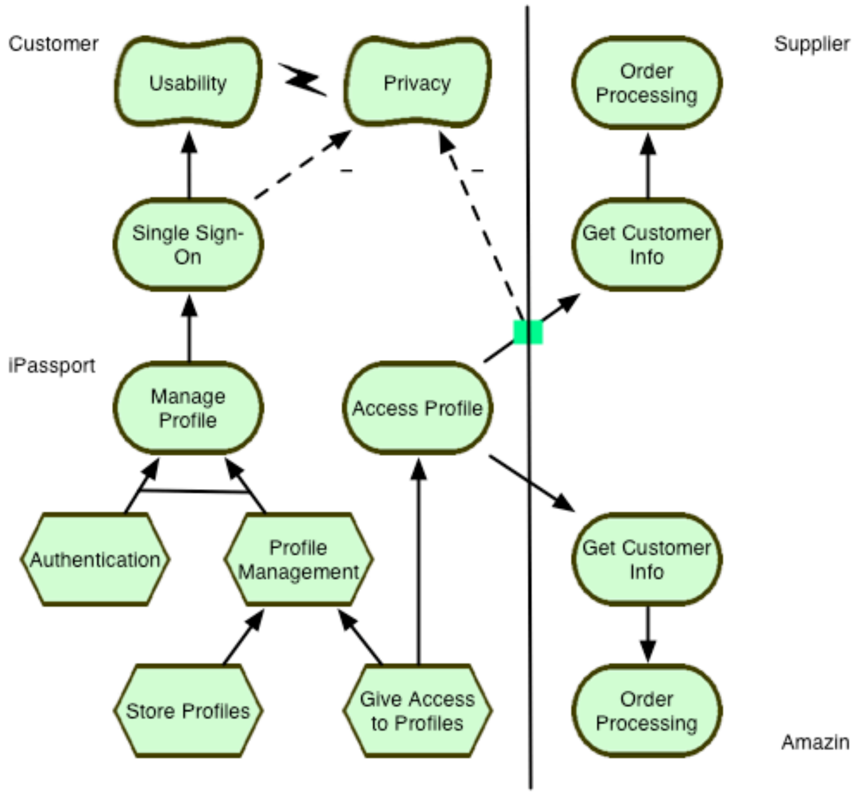

Figure 19. Example of a goal conflict in the Amazin application

This diagram adds two new notational elements. Bars, which can be vertical or horizontal, indicate boundaries between actors. They are only used for emphasis, in a manner similar to swimlanes in activity diagrams. Here a bar separates iPassport from Supplier and Amazin. The square on a contribution or correlation link is used to highlight the link. It can be used to draw attention to a link that is itself the source of another link. Again, highlighting is used for emphasis and not strictly required by our notation. The link from Access Profile to the Get Customer Info goal of the Supplier represents that any Supplier can gain access to the user's profile via the Access Profile service. This link is highlighted, and itself linked to the Privacy goal to indicate that providing uncontrolled access to the user's profile has the side-effect of violating their Privacy concerns.

6.2 Resource Contention

When Amazin invokes the Order Processing 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, resulting in an order processing loop. Figure 20 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 will hurt the performance of both servers as as side-effect. This conflict is a feature interaction between the Order Processing services offered by Amazin and its Suppliers.

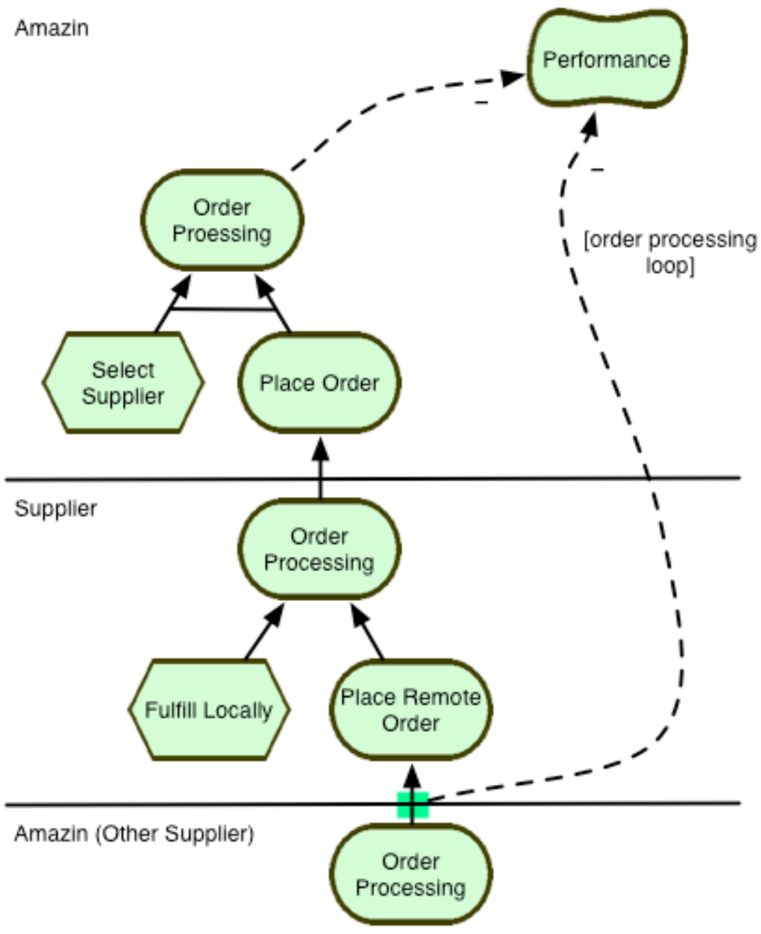

Figure 20. Example of an interaction due to resource contention in the Amazin application

6.3 Policy Conflict

A Customer might benefit from more than one type of discount, for example, based on membership and bulk purchases. This leads to a possible conflict between policies as retrieved through Access Profile (either various policies within the same profile, or across local profiles stored with the various Suppliers), and to errors in Order Processing. There needs to be a way to specify that one discounting policy should override the others, or whether some or all of the policies must be applied. In case the implicit meta-policy is to apply only one discounting policy, then the invocation order among multiple calls to the Access Profile service can become another related issue, unless a complete priority list is established among the various policies. However, establishing such a list is generally impractical in an open and distributed domain such as web service-based e-commerce, where suppliers and policies can be added dynamically.

6.4 Invocation Order

There is a potential conflict between Payment Processing and Order Processing, or Payment Processing and Delivery due to timing errors, as shown in Figure 21. 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 example, when the customer cancels their order, it could be that the payment still gets processed, but order shipment is aborted. The reason is that the cancellation request was sent just before payment started, but arrived after the payment process has proceeded. Technically, such problems can be avoided through transaction management, but this requires that the problem has been properly anticipated in the first place.

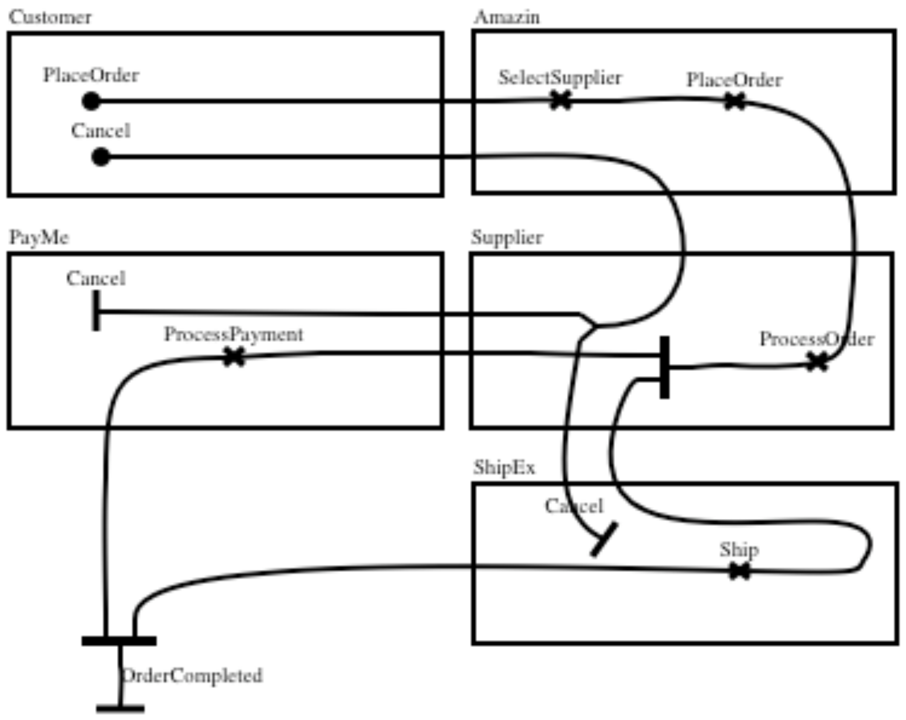

Figure 21. Example of an interaction due to invocation order in the Amazin application

7 RELATED WORK

Goal-oriented analysis as introduced by (Mylopoulos et al., 1999; Chung et al., 2000), and supported by the GRL notation (GRL, 2003), is the primary analysis method used in this paper. This method has been applied to early requirements engineering in a number of domains, including telecommunications. However, to our knowledge, this is the first paper describing the use of goal graphs for modeling and reasoning about feature interactions.

Prior work on using UCMs for modeling features, and detecting feature interactions is reported in (Amyot 2000). That paper describes a scenario-based approach to generating validation test suites, and feature interaction detection by identifying scenarios with overlapping preconditions. Although the approach has only been applied to telecommunication systems, we don't see why it could not be applied to detect similar interactions between web services. However, non-functional feature interactions, and deployment issues are not considered.

An early version of some of the ideas in this paper is contained in (Weiss et al., 2004). This paper differs from the earlier paper in a number of significant aspects. It introduces a new case study, makes use of the Use Case Map notation to analyse deployment and ownership issues, and provides a much more comprehensive empirical analysis of feature interactions in web services. It also presents current results towards a classification of feature interactions in web services, and illustrates it with an example of a typical e-commerce application.

8 CONCLUSION

In this paper, we proposed an approach for modeling undesirable interactions between web services as feature interactions, and their detection. Our approach is unique in its use of goal graphs from goal-oriented analysis (Goal-Oriented Requirements Language, GRL), and to reason about feature interactions, and scenario models (Use Case Maps, UCMs) for reasoning about the deployment of functional features. Our paper includes an empirical analysis of several feature interactions. In our discussion of the case studies, we identified a number of reasons for undesirable interactions among web services, including goal conflicts, deployment and ownership, as well as information hiding issues, as well as strategies for resolving such interactions.

The combination of GRL and UCMs supports our approach well. The first two steps outlined in Section 3, modeling features, and analyzing these models for goal conflicts, can be achieved using the goal graph concept from GRL. Goal graphs allow us to represent features, and to reason about conflicts between them. The third step, resolving interactions, can be supported by using GRL in combination with UCMs. GRL is suitable for reasoning about refactorings of the goal graph. UCM models allow us to explore the different alternatives suggested by the GRL models. They are particularly suitable for analyzing dependencies between, and changes to the deployment and ownership of functional features. They can also be used to reason about intention-disclosing negotiation protocols for resolving interactions due to information hiding.

We are currently working on a benchmark for feature interactions in web services to encourage the comparison of different modeling and resolution approaches. We also study the formalizations of goal graph refactorings, for example, using graph rewriting techniques. In other work we are exploring some of the open research questions triggered by this research: what parallels are there between non-functional features and aspects in Aspect-Oriented Programming (AOP), between goal conflicts and the interaction of aspects; and what kind of architecture can support the runtime management of feature interactions in aspect- and service-oriented architectures.

REFERENCES

AgentCities Lisboa Node, Restaurant Finder Service, http://agentcities.we-b-mind.org/services/RFService Description.htm, last accessed September 2004.

Amyot, D., Charfi, L., et al (2000), Feature description and feature interaction analysis with Use Case Maps and LOTOS, International Workshop on Feature Interactions in Telecommunications and Software Systems (FIW), IOS, 274--289.

Amyot, D. (2003), Introduction to the User Requirements Notation: Learning by Example, Computer Networks, 42(3), 285-301.

Arsanjani, A. (2002), Towards a Pattern Language for Web Services Architecture, Pattern Languages of Programming (PLoP).

Chung, L. (1991), Representation and Utilization of Non-Functional Requirements for Information System Design, Conference on Advanced Information Systems (CAiSE), LNCS, 5-30, Springer.

Chung, L., Nixon, B., Yu, E., and Mylopoulos, J. (2000), *Non-Functional Requirements in Software Engineering*, Kluwer.

Constantinescu, I., Willmott, S., and Faltings, B. (2002), Abstract Behavior Representations for Service Integration, AAMAS Workshop on AgentCities: Challenges in Open Agent Environments.

Google, Google Web APIs, http://www.google.com/apis/reference.html, 2003.

GRL, http://www.cs.toronto.edu/km/ GRL, last accessed Aug 2003.

Hailpern, B., and Tarr, P. (2001), Software Engineering for Web Services: A Focus on Separation of Concerns, OOPSLA Workshop on Object-Oriented Web Services, available as IBM Research Report, RC 22184.

Liberty Alliance, http://www.projectliberty.org, last accessed Aug 2003.

Microsoft, Microsoft .NET Passport for Developers, http://www.microsoft.com/net/services/ passport/developer.asp, last revised May 2003.

Montaner, M., Lopez, B., and de la Rosa, J. L. (2002), Opinion-Based Filtering Through Trust, Workshop on Cooperative Information Agents (CIA), LNAI 2446, 164-178, Springer.

Mylopoulos, J., Chung, L., and Yu, E. (1999), From Object-Oriented to Goal-Oriented Requirements Analysis, Communications of the ACM, 42:1, 31-37.

O'Sullivan, J., Edmond, D., and ter Hofstede (2002), A., What's in a Service? Towards Accurate Description of Non-Functional Service Properties, Distributed and Parallel Databases, 12, 117-133, Kluwer.

Perkins, E. (Feb 5, 2002), Passing Passport, Meta Group, Tech Update Newsletter, ZDNet.

Pulvermüller, E., Speck, A., et al (2001), Feature Interaction in Composed Systems, ECOOP Workshop on Feature Interactions in Composed Systems.

Ryman, A. (2003), Understanding Web Services, http://www.software.ibm.com/wsdd/library/techarticles/ 0307_ryman/ryman.html.

Snell, J., Tidwell, D., and Kulchenko, P. (2002), *Programming Web Services with SOAP*, O'Reilly.

Sun, Sun ONE Studio 5: J2EE Application Tutorial, http://developers.sun.com/tools/javatools/ documentation/s1s5/diningguide.pdf, 2003.

UDDI (Universal Description, Discovery, and Integration of Web Services), http://www.uddi.org, last accessed Aug 2003.

URN (User Requirements Notation), http://www.usecasemaps.org/urn, last accessed Aug 2003.

Utas, G. (2001), A Pattern Language of Feature Interactions, in: Rising, L., *Design Patterns in Communication Software*, Cambridge University Press,

Velthuijsen, H. (1993), Distributed Artificial Intelligence for Runtime Feature Interaction Resolution, IEEE Computer, 45-55.

W3C, Web Services Architecture Usage Scenarios, S030, http://www.w3.org/TR/2002/WD-ws-arch-scenarios-20020730, 2002.

W3C, Platform for Privacy Protection (P3P) Project, http://www.w3.org/P3P, last revised Aug 2003.

Weiss, M., Esfandiari, B. (2004), On Feature Interactions in Web Services, Proceedings of the 2nd International Conference on Web Services (ICWS), IEEE.

Zacharia, G., Moukas, A., and Maes, P. (1999), Collaborative Reputation Mechanisms in Electronic Marketplaces, Hawaii International Conference on System Sciences (HICSS), 8026, IEEE.

ABOUT THE AUTHORS

Dr. Michael Weiss is an Assistant Professor at Carleton University, which he joined in 2000 after after spending five years in industry following his PhD in Computer Science in 1993 (University of Mannheim, Germany). In particular, he lead the Advanced Applications group within the Strategic Technology group of Mitel Corporation. His research interests include web services, software architecture and patterns, business model design, and open source.

Dr. Babak Esfandiari is also an Assistant Professor at Carleton University. He obtained his PhD in Computer Science in 1998 (University of Montpellier, France) then worked for two years at Mitel Corporation as a software engineer before joining Carleton in 2000. His research interests include agent technology, network computing and object-oriented design.