Towards Matching of Service Feature Models based on Linear Logic

Muhammad Naeem  
Dept. of Comp. Sci.  
University of Leicester  
mn105@mcs.le.ac.uk

Reiko Heckel  
Dept. of Comp. Sci.  
University of Leicester  
reiko@mcs.le.ac.uk

ABSTRACT
When matching services, we are caught in the tradeoff between precise specifications required to make dynamic binding safe and the lack of flexibility that comes with this precision. We propose to use feature modelling techniques to specify the variability of provided and required services, thus increasing the flexibility of the matching process. In particular, we address the matching of one required against several provided specifications, distinguishing between static matching, where all services are known in advance, and dynamic matching, where one provided service is matched at a time. After analysing the requirements for matching feature models in this context, we use an interpretation in linear logic to analyse their use in service specifications and matching.

Categories and Subject Descriptors
D.2 [Software]: Software Engineering; D.2.8 [Software Product Lines]: Feature Models—Matching of Service Feature Models

General Terms
Theory

Keywords
Feature models, Partial matching, Service composition

1. INTRODUCTION
The vision of Service-oriented Architectures [6] is to support flexible on-demand interactions in a market place of application components where requestors’ requirements are matched against providers’ offers. Automation of dynamic discovery, selection and binding of services continues to pose major challenges. One of them is the level of flexibility required to find a service to specific requirements. Any automated selection requires specification not only of the signatures and data types of the services, but also of their actions and protocols. A more detailed description will, however, be less likely to be matched by any existing service.

For example, a requestor while planning to arrange a trip to attend a conference, may not only be interested in booking a flight and a hotel but also to get tickets for a play and to have dinner at a well-known restaurant. No travel agency takes care of all these activities. In a scenario where a single offer cannot satisfy all requirements, there are two solutions: To compromise on the requirements, e.g., by abandoning cultural activities, or use more than one provider, e.g., by booking directly with the restaurant.

In this paper, we propose the use of feature models to capture both the requestor’s and provider’s flexibility distinguishing, among other aspects, necessary and optional requirements. First, we discuss the challenges of matching a feature model specifying a requestor’s goal against models that describe providers’ offers. The analysis leads us to the conclusion that feature models should be interpreted as linear logic formulas, and that deduction in linear logic provides the semantics for matching. The problem of matching one goal against several offers is addressed both statically, where all offers are known in advance, and dynamically, where one offer is chosen and matched at a time and a remainder goal is constructed for matching with future offers.

The rest of the paper is organised as follows: Section 2 provides some background about feature models. Section 3 states requirements for matching feature models specifying services. The interpretation of feature models in linear logic is detailed in Section 4. Section 5 addresses static and dynamic matching. We discuss related work in Section 6 and conclude in Section 7. Section 8 outlines the future work.

2. SERVICE FEATURE MODELS
Usually, feature models represent hierarchies of common and variable features in software product lines [9][10]. Here we use them to describe variability of services while considering the notations of Czarnecki and Eisenecker [3]. A service feature model is given with respect to an alphabet declaring the vocabulary (set of feature names) that can be used in the model. This allows us to use different vocabularies to describe services based on different ontologies and to handle these vocabularies explicitly in the matching. By $M = \langle \alpha, FD \rangle$ we refer to the feature model where $FD$ is a feature diagram [3], see Fig. 1 over the alphabet $\alpha$. An instance of $M$ is a subset of the alphabet that is consistent with the constraints specified by $M$. For example, a
valid purchase from the travel agent shown in Fig. 1 must obey the following rules: i) If a feature is selected, its parent must also be selected. The root feature is always selected. ii) If a parent is selected, all the mandatory features of its And group are selected. iii) If a parent is selected, exactly one feature of its Alternative group must be selected. iv) If a parent is selected, at least one feature of its Or group must be selected. v) If two features are related by a constraints (shown as a dashed arrow), the target feature is included whenever the source is (shown as a dashed arrow), the target feature is included.

Figure 1: Feature diagram of an On-line Travel Agent Service

A declarative way to define the notion of instance is by means of propositional logic [1, 5]. For example, \((\text{T A} \leftrightarrow \text{Res}) \land (\text{Res} \leftrightarrow (\text{H} \lor \text{F})) \land (\text{Tr} \rightarrow \text{T A}) \land (\text{Tr} \rightarrow \text{H}) \land (\text{T A} \leftrightarrow \text{Pay}) \land (\text{Pay} \leftrightarrow ((\text{CC} \lor \text{BT}) \land \neg(\text{CC} \land \text{BT})))\) is a propositional equivalent of the feature diagram shown in Fig. 1.

Solutions to this formula represent precisely all instances of the feature diagram.

3. MATCHMAKING CHALLENGES

To verify that a collection of services offered jointly satisfy the goals of a requestor, we have to match their specifications [9]. Specifying goals and offers by means of feature models, we can take account of the variability allowed for and required by providers and requestors. Below we are discussing the challenges created by the matching of service feature models.

3.1 Resource vs. Shareable Feature

When a requestor’s goal is to be satisfied by a combination of multiple offers, the question may arise, whether a specific feature can be satisfied more than once. Let us consider a scenario where a requestor wants to book flight, hotel, and transport as shown in Fig. 2(a). This requestor finds two offers shown in Fig. 2(b) and 2(c). The first offer provides reservations of hotel and transport whereas the second one allows to book hotel and flight, but both describe hotel reservation as a mandatory. The objective is to reserve one hotel room.

\(^1\)We can also have excludes constraint shown by a dashed line with arrows at both ends, which means that source and target cannot be chosen in a single instance.

\(^2\)For brevity, when denoting features in instances (or logical formulas), we use the underlined letters in the feature name only.

Figure 2: Requestor and provider feature models

The classical interpretation of feature models in propositional logic is unable to distinguish between single or multiple occurrences of the same feature. Formalising the match by implication \(\text{Prov}_1 \land \text{Prov}_2 \Rightarrow \text{Req}\) we would be misled into believing that the two offers jointly satisfy the requestor’s goal, resulting in booking the hotel room twice. \(^3\)

Formally, this is a consequence of the idempotence of conjunction in propositional logic, i.e., \(A \land A = A\). Therefore, two hotel reservations are indistinguishable from one. More generally, every feature is interpreted as shareable, i.e., it can be satisfied or used multiple times. For a meaningful notion of matching of multiple feature models we have to distinguish such features from resource-like ones that can be used or satisfied once. For example, the reservations in Fig. 2(a) should be marked as resources so that no combination of offers providing more than one hotel reservation can satisfy the requestor’s goal.

Unfortunately, the distinction between resource and shareable feature neither exists at the level of feature models nor in classical logic. Linear logic can differentiate between single and multiple instances of linear propositions. We will discuss this distinction more formally in Section 3.

3.2 Partial Matching

The problem of matching one goal by multiple offers is simplified if all offers are on the table at the time of matching, i.e., if all offers are matched at design time or before the service is invoked. With long-running transactions in a dynamic market, we cannot be sure that the services on offer at the time of matching are still available when they are actually required, or if they still represent the best choices. Therefore we consider a dynamic approach where matching is assumed to happen at runtime. In this case we have to match and bind to services incrementally, one service at a time. This means that certain required features remain open while others are already satisfied by offers matched so far.

In the incremental approach, a partial match satisfies some of the required features. The remaining features are captured in a remainder goal which provides the starting point for further rounds of partial matching. We discuss both static and dynamic matching of offers in Section 3.

3.3 Your Choice or Mine

With requestor and provider being independent entities, there are different ways to resolve alternatives. For example, if a

\(^3\)For the moment we gloss over the differences in naming of features in requestor and provider diagrams. These will be relevant later when discussing mapping of alphabets.
customer looking for a room with sea view finds a hotel offering rooms with sea view or facing inland, the requestor is sure to be satisfied if they get to choose the type of room. If the rooms are assigned by the hotel based on availability, satisfaction is not guaranteed.

Consider, for example, the feature diagrams for provider and requestor in Fig. 3. If the provider chooses between B and C, there is no guarantee that the requestor’s goal C is satisfied. In terms of propositional logic, if the provider makes the choice the offer is rejected because \( B \lor C \rightarrow B \) is not valid. If the requestor makes the choice we only require satisfiability. Since the formula is satisfiable (for \( B = \text{true} \)), the offer can be accepted.

More generally we would prefer to be flexible, allowing the model to declare for each alternative set of features if the choice will be made by the requestor or the provider. For example, while rooms may be allocated based on availability by the hotel, the requestor could have the choice between a continental or English breakfast, thus combining both cases in a single model. Such a fine-grained distinction is not possible in classical logic.

### 3.4 Keeping my Options

Matching of requestor with provider models and instantiation of the selected services are seen as separate steps. For example, a requestor planning a dinner with friends at a restaurant may be interested in the option of vegetarian meals but will let the guests choose for themselves once they have arrived. An optional feature or alternative in the requestor model should not be disregarded in the matching but matched by a corresponding option or alternative, i.e., a restaurant that can provide both vegetarian and non-vegetarian food.

This is impossible based on the simpler approach where requirements are represented by instances and matching corresponds to checking if the instance is permissible based on the feature model specifying the offered service. Instead, capturing requirements about variability both requestor and provider are represented by feature models.

### 3.5 Mapping of Feature Names

If requestor and provider feature models are developed separately, they may use different alphabets. To be able to relate models using different alphabets we have to provide an explicit mapping as part of the matching.

For example, \( H_R \) of Fig. 2(a), \( H_{P1} \) of Fig. 2(b), and \( H_{P2} \) of Fig. 2(c) represent the same feature, i.e., reservation of a hotel room, but use different names. In this case, \( H_{P1} \) of Fig. 2(b) and \( H_R \) of Fig. 2(a) are mapped with the element representing hotel reservation in shared ontology which leads to the mapping of \( H_{P1} \) with \( H_R \).

Such a mapping will only be semantically meaningful if both alphabets can be interpreted over a common ontology, such that the mapping is compatible with these interpretations. For example, there is the possibility of describing features at different levels of abstraction: A requestor model may talk about payments in general, while the a provider allows bank transfer or credit card. In this case, in order for a mapping of these features to be semantically compatible, in the shared ontology payment should be a supertype of credit card and bank transfer.

### 3.6 I Did Not Order That

A feature model may not make explicit statements about all the features in the alphabet. The classical interpretation of a logical specification would assume it to be incomplete, so that more information can be added by a refinement of the specification. From the requestor’s perspective, however, this would mean that all features in the alphabet that are not ruled out by the model are permissible too, i.e., we may end up with more features than we bargained for.

In classical logic the closed-world assumption addresses this problem assuming that any statement that is not contained in or derivable from the specification is false. That means, requestor specifications should be seen as complete so that they cannot be matched by providers adding features not explicitly approved.

### 4. FEATURE MODELS IN LINEAR LOGIC

Based on the discussion above we suggest that linear logic is more appropriate than propositional logic as a semantic interpretation of feature diagrams modelling services. Formulas in classical logic deal with sentences that are either true or false. Linear logic formulas can be interpreted as statements about resources. For instance, \( A \otimes B \) states the selection of both resources A and B. Similarly, we are able to distinguish single from multiple occurrences of resources, i.e., \( A \otimes A \neq A \) allows to count the number of occurrences. The !-modality, used to declare a resource unbounded, states that \( !A \) can be shared. This allows to distinguish shareable features and resources.

In order to mark the distinction in feature diagrams we use a grey shade for shareable features. For example, the feature diagrams of Fig. 2 are revised in the light of this distinction in Fig. 4. Their root nodes are shareable because the requestor has to be willing to deal with more than one provider, while providers have to accept non-exclusive dealing with the requestor. This applies only to the shaded nodes themselves and not to their children. As discussed earlier, reservations should only be made once, hence all other nodes are resources.

With regards to who gets to make the choice in case of alternatives, for the time being we limit ourselves to the “customer-friendly” case where the choice is made by the requestor. In linear logic, this is the semantics of the additive conjunction “&” operator. We use it to encode optional features \( A \& A^+ \) as choice between selection and non-selection as well as for representing alternatives \( A \& B \) (not shown in Fig. 2 or Fig. 4).

If instead the provider were to keep the choice, we have to
Table 1: Mapping of Feature in Linear Logic

<table>
<thead>
<tr>
<th>Rules</th>
<th>Linear equivalence of different feature types</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>( LF(\Box) = x )</td>
<td>A feature maps to a linear variable.</td>
</tr>
<tr>
<td>2</td>
<td>( LF(\Box \rightarrow X) = x \otimes x \otimes (x \rightarrow LF(X)) )</td>
<td>Sub-tree ( X ) is <strong>mandatory</strong>, i.e., must be present if feature ( x ) is. In this case, we can derive ( x \otimes LF(X) ).</td>
</tr>
<tr>
<td>3</td>
<td>( LF(\Box \rightarrow X) = x \otimes x \otimes (x \rightarrow (LF(X) &amp; LF(X)^+)) )</td>
<td>( X ) is <strong>optional</strong>, expressed by choice between ( LF(X) ) and ( LF(X)^+ ). If ( x ) is selected we can derive both ( LF(X) ) and ( LF(X)^+ ).</td>
</tr>
<tr>
<td>4</td>
<td>( LF(\Box) = !x )</td>
<td>Feature ( x ) is <strong>shareable</strong>, behaving like a classical proposition. From (!x) we can derive any number of occurrences ( x \otimes \cdots \otimes x ).</td>
</tr>
<tr>
<td>5</td>
<td>( LF(\Box \bigoplus X_1 \cdots X_n) = x \otimes x \otimes (x \rightarrow ((LF(X_1) \otimes LF(X_2))^+ \otimes LF(X_3)^+ \otimes \cdots LF(X_n)^+ &amp; \ldots LF(X_2) \otimes LF(X_1)^+ \otimes LF(X_3)^+ \otimes \cdots LF(X_n)^+) &amp; \ldots LF(X_n) \otimes LF(X_1)^+ \otimes LF(X_2)^+ \otimes \cdots LF(X_n)^+)) )</td>
<td>The exclusive OR of subtrees ( X_1, X_2, \ldots, X_n ) is expressed by the choice of between selecting any ( X_i ), while deselecting all others. All of these choices are derivable from ( x ).</td>
</tr>
<tr>
<td>6</td>
<td>( LF(\Box \bigcap X_1 \cdots X_n) = x \otimes \ldots \otimes (x \rightarrow (LF(X_1) \otimes LF(X_2)^+ \otimes LF(X_3)^+ \otimes \cdots LF(X_n)^+) &amp; \ldots LF(X_n) \otimes LF(X_1)^+ \otimes LF(X_2)^+ \otimes \cdots LF(X_n)^+) &amp; \ldots LF(X_n) \otimes LF(X_1)^+ \otimes LF(X_2)^+ \otimes \cdots LF(X_n)^+) )</td>
<td>Subtrees ( X_1, X_2, \ldots, X_n ) are part of an <strong>Or</strong> group. The mapping is as above, but allowing for a choice between all subsets of features, deselecting their respective complements.</td>
</tr>
<tr>
<td>7</td>
<td>( LF(\Box \bigotimes X_1 \cdots X_n) = \bigotimes_{i=1}^n (x \otimes x \otimes (x \rightarrow LF(X_i))) )</td>
<td>The <strong>And</strong> group combines a set of mandatory subtrees ( X_i ), which must all be present if ( x ) is.</td>
</tr>
</tbody>
</table>
use the additive disjunction “⊕” operator. We refrain from formalising this for the moment because it would require a duplication of most of the syntactic elements of feature diagrams, adding “internal choice” versions for the existing “external choice” ones.

Table 1 shows the rules by which we map feature models $M = (\alpha, FD)$ into linear logic formulas $LF(M)$. Rule 1 says that the root is always part of each interpretation, followed by the formulas obtained from its children. Rules 2 and 3 deal with mandatory and optional sub-features, using linear implication $a \rightarrow b$ to express that $b$ has to be selected when $a$ is. In contrast to propositional implication $a \rightarrow b$, $a$ is consumed in the process. Thus, in order to keep intermediate nodes of the tree as possible instances we provide another copy of the parent feature. For example, $a \otimes a \otimes (a \rightarrow b)$ represents a feature diagram with root $a$ and mandatory sub-feature $b$, entailing $a \otimes b$. As anticipated, shareable features are encoded using $!$ (Rule 4) while Alternatives and Or are based on suitable combinations of $\&$ as a duplication of most of the syntactic elements of feature models in Fig. 4(a, b, and c). It should be noted that “Req”, being shareable, is mapped to both functions, while everything else is mapped to only one. The fact that both functions are jointly surjective, reaching every requestor feature, is typical for a case like ours, where all requestor features are mandatory.

The set of all linear instances is derivable for a feature model using deduction in linear logic. For a feature model $(\alpha, FD)$, $LF(\alpha, FD) \vdash_{LL} \bigwedge_{i=1}^{n} LI_i$ represents the deduction of the choice between all linear instances $LI_i$ of the model.

5. MATCHING OF FEATURE MODELS

In order to match service feature models, they need to be given over a common alphabet. This is achieved in this section by assuming a mapping relating the alphabets of different models based on which a translation of models and formulas can be given. Following that we study the matching both in its static and dynamic (incremental) variant.

5.1 Mapping of Alphabets

We assume a mapping $m$ to capture the relation between the alphabets of requestor and provider feature models. Given feature models $R = (\alpha_R, FD_R)$ for the requestor and $P_i = (\alpha_{P_i}, FD_{P_i})$ for each provider, the mappings are total functions

$$m_i : \alpha_{P_i} \rightarrow \alpha_R \cup \{\perp\}$$

The function maps feature names from the provider’s to the requestor’s alphabet or bottom $\perp$, the unit of multiplicative disjunction, analogous to false in propositional logic. That means, provider features not mapped to requestor ones are disregarded.

<table>
<thead>
<tr>
<th>Provider 1 ($P_1$)</th>
<th>Provider 2 ($P_2$)</th>
<th>Requestor ($R$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{Res}_{P1}$</td>
<td>$\text{Res}_{P2}$</td>
<td>$\text{Res}$</td>
</tr>
<tr>
<td>$\text{Ht}_{P1}$</td>
<td>$\text{Ht}_{P2}$</td>
<td>$\text{Ht}$</td>
</tr>
<tr>
<td>$\text{Fl}_{P1}$</td>
<td>$\text{Fl}_{P2}$</td>
<td>$\text{Fl}$</td>
</tr>
<tr>
<td>$\text{Tr}_{P1}$</td>
<td>$\text{Tr}_{P2}$</td>
<td>$\text{Tr}$</td>
</tr>
<tr>
<td>$\text{Req}$</td>
<td>$\perp$</td>
<td>$\perp$</td>
</tr>
</tbody>
</table>

For example, Table 2 shows the mappings of the alphabets of the feature models in Fig. 4(a, b, and c). It should be noted that “Req”, being shareable, is mapped to both functions, while everything else is mapped to only one. The fact that both functions are jointly surjective, reaching every requestor feature, is typical for a case like ours, where all requestor features are mandatory.

The mapping is used to substitute feature names. This renaming can lead to features being replaced by $\perp$. If such a
substitution is applied to a formula $F = a \otimes b$, e.g., under $m(a) = a'$ and $m(b) = \bot$, the result of applying $m$ to $F$ is $m(F) = a' \otimes \bot$. Note that, in particular, $a'$ cannot be derived from $a' \otimes \bot$. Using the mappings in Table 2 the linear equivalents of the provider diagrams of Fig. 4 (b, c) are shown in Formulas (2) and (3).

\[
!Req \otimes (!Req \rightarrow (Res_H \otimes (Res_H \otimes Res_H \rightarrow H_R))) \otimes (!Req \rightarrow (Res_{T_1} \otimes (Res_{T_1} \otimes Res_{T_1} \rightarrow T_{R_1})))
\] (2)

\[
!Req \otimes (!Req \rightarrow (\bot \otimes (\bot \otimes \bot \rightarrow \bot))) \otimes (!Req \rightarrow (Res_F \otimes Res_F \rightarrow F_R)))
\] (3)

Note that $Res_{H_1}$ and $Ht_{P_2}$ have been replaced by $\bot$ in Formula (3), representing the fact that the second provider’s hotel offer remains unused.

### 5.2 Static Matching

A single offer satisfies a requestor’s goal (over the same alphabet) if the linear formula of the goal can be derived from that of the offer i.e., if $LF(P) \vdash LF(R)$. If, as is usually the case, we want to match one goal against several offers, we use multiplicative conjunction to combine the provider formulas taking part in the match, i.e.,

\[
\bigotimes_{i=1}^{n} LF(P_i) \vdash LF(R)
\]

where $n$ is the number of offers. In our example we are interested in verifying that $LF(P_1) \otimes LF(P_2) = [2] \otimes [3] \vdash [1] = LF(R)$, where Formulas (2) and (3) are obtained from applying the mappings $m_l$ to the linear formulas derived from the two provider models.

Let us analyse this choice in view of our requirements. Due to $A \otimes \bot \not\vdash A$, we cannot derive the requestor specification of Fig. 4(a) from the combined offers shown in Fig. 4 i.e., $LF(P_1) \otimes LF(P_2) \not\vdash LF(R)$. This is consistent with our intuition about the scenario. If, instead, we consider the hotel reservation of the second provider as an optional feature, the combined effect of both offers satisfies the goal, i.e., $LF(P_1) \otimes LF(P_2) \vdash LF(R)$ because $A \otimes (\bot \& \bot^\circ) \vdash A$.

In analogy to the discussion in Section 3.4 shareable features requested must be matched by shareable ones provided. In terms of linear logic this is expressed as $A \not\vdash !A$ where $A$ is shareable in the requirement (conclusion) and a resource in the provision (premise), i.e., a resource can not satisfy a requirement for a shareable feature. In the case of a single provider this guarantees that any expectations of unlimited supply are actually matched by the offer. For example, in an all-inclusive holiday booking no limit is imposed on the number of excursions taken. A requestor can specify such a requirement as a shareable feature.

If a shareable feature is used to combine offers of several providers, such as payment by a single credit card for services provided by different travel agencies, only offers that accept shareable payments should be able to satisfy the requestor’s demand.

### 5.3 Towards Dynamic Matching

If not all the providers’ offers are known in advance, matching has to be incremental. In this case, after matching the selected offer with the goal, we are interested in extracting the remaining, as yet unfulfilled requirements. These are matched with the next offer until all requirements are satisfied or we run out of offers. We assume that the selected offer contributes towards the goal, thus guaranteeing progress.

For example, consider the scenario shown in Fig. 4. We select the offer in Fig. 4(b). After matching it with the requestor’s goal we extract the unfulfilled requirement of a flight reservation. This can then be matched with the next available offer providing the flight reservation.

To implement this idea, we have to characterise the remainder of a goal with respect to a partial match. In logical terms, if $Rem, LF(P), LF(R)$ show the linear formulas for the remainder, provider, and requestor models, respectively, we are interested in the weakest formula $Rem$ such that

\[
Rem \vdash LF(P) \rightarrow LF(R)
\]

If such a remainder $Rem$ does not exist, we have to reject the offer $P$ because it cannot be used as a step towards a successful completion of the match. For example, a mandatory feature of the offer may be in contradiction with the requestor’s goal.

This gives rise to the following process of partial matching for a requestor’s goal $R$.

1. Select available offer $P$ equipped with a mapping $m$ to $R$’s alphabet. Terminate the procedure if no such offer is available.

2. Apply mapping $m$ to $P$ to obtain $m(P)$.

3. Derive mapping $m$ to $P$ to obtain $m(P)$.

4. If remainder $Rem$ is empty, use $Rem$ as the new goal $R$ and go back to step 1.

Intuitively, the remainder $Rem$ consists of all features not satisfied by the provider as well as the shareable features of the requestor goal $R$.

Let us consider in more detail the scenario shown in Fig. 4 for the dynamic matching of the two offers with the goal. As outlined above, we select an available offer (starting with Fig. 4(b)) and apply the mapping to translate it to the requestor’s alphabet. We generate the remainder $Rem$, which is $\langle Req \otimes (Req \rightarrow (Res_F \otimes (Res_F \otimes Res_F \rightarrow F_R))) \rangle$. The alphabet of the remainder consists of all features which exist in the requestor goal, but are not in the offer, along with the shareable features of the requestor goal, i.e., $\langle Req, Res_F, F_R \rangle$. Now considering the remainder as the new goal, we loop back...
to step 1 and select the next offer (Fig. 4(c)). After applying the alphabet mapping we observe that construction of the remainder is not possible because it would require us to select $Res_{H1}$ and $H_{P2}$. However, these features are no longer available on the requestor’s side. Therefore we reject this offer and go back to step 1 again. Since we do not have any available offer left, the procedure is terminated unsuccessfully.

If, instead, we modify the second offer by marking $Res_{H1}$ and $H_{P2}$ as an optional features, we can derive an empty remainder. That means, the offer can be selected and the procedure terminates successfully.

We can now derive the original goal from the multiplicative conjunction of all selected offers. The result shows that the goal is satisfied by the combined offers in Fig. 4 if the second offer designates $Res_{H1}$ and $H_{P2}$ as optional features.

6. RELATED WORK

In [13] we have discussed the matching of individual offers with requestors’ goal based on the interpretation of feature diagrams in propositional logic. As demonstrated in the scenario of Fig. 4 the need to distinguish shareable from non-shareable features (resources) arises only when matching a goal with several offers.

Czarnecki et al. [4] have used cardinalities in feature diagrams. The interpretation in propositional logic uses indexed propositions $a_1 \land \ldots \land a_n$ to represent multiple occurrence of a feature $a$. Apart from the fact that this use of names to encode multiplicity information is not reflected in the semantics and calculus of propositional logic, it does not solve our problem: The use of such clonable features in our scenario does not guarantee that a requestor is given exactly one hotel reservation. For example, if we use cardinality 1..1 for all parent-child relations, which is the closest we can get to considering them as resources, the feature models in Fig. 4(b), (c) would still satisfy the requestor goal in Fig. 4(a). Again, the problem here is matching multiple providers with overlapping offers.

Shareable features are also different from clonable ones in the sense that being shareable is a property of a node that is not inherited by its children. For example, in Fig. 4(a), the root node $Req$ is shareable while the rest of the nodes in the tree represent resources. This means, $Req$ can be matched multiple times (by several offers) while the features representing individual reservations have to be matched exactly once.

While there does not appear to be work relating feature models and linear logic, a number of authors have explored the use of feature models in the context of service-oriented architectures. Their focus has been on improving reuse in a feature-oriented product line-engineering approach [12] based on feature analysis technique to identify services; on developing domain-specific services and optimising discovery based on customising preexisting results [15]; and on the overall methodology of variation-oriented engineering of services, accounting for variations in service development, composition, and configuration [15]. Our approach is more specifically targeted at the matching of service providers’ descriptions against requestor’s requirements. Matching multiple descriptions, variability may arise from how they are combined rather than the individual services themselves. More generally, we provide a new semantic interpretation of feature modelling motivated by service matching, but not necessarily limited to this domain.

The use of linear logic in the context of services has been suggested in [13], focussing on composition rather than matching. A proof in intuitionistic linear logic is constructed deriving the global pre/post condition of the composition from pre/postconditions of its constituent services. The proof itself is taken as a template for the composition. The interpretation of a linear formula in this approach is computational, with linear implication representing state transitions brought about by the invocation of elementary services. In our case a formula represents the resources sought after or on offer and implication represents a step in the top-down instantiation of the corresponding feature model.

7. DISCUSSION

In this section, we evaluate how the challenges elaborated in Section 3 are realised by our approach and discuss open problems and questions.

Resource vs. Shareable Features: Linear logic allows the distinction between shareable and non-shareable features, interpreting linear variables as resources and providing the $!$-modality to distinguish shareable features. We provide a mapping of (extended) feature diagrams to linear logic such that matching of service specifications is represented by deduction.

Partial Matching: If an offer satisfies only partially the goal of a requestor, a remainder of the goal is constructed to be matched against offers discovered in the future. This incremental approach to matching is suitable for dynamic binding because it allows to delay the match until the time it is needed.

Your Choice or Mine: Linear logic allows to distinguish between choices made by the provider or requestor of the service. In this paper we only consider the more common case where all choices are made by the requestor. The more general approach requires a significant syntactic extension of feature diagrams which we leave for future work.

Keeping my Options: Optional features, alternatives decided by the requestor and shareable features can only be matched by features of similar flexibility. That means, when matching feature models, the level of flexibility does not decrease. Abandoning one’s options would be possible, for example, if the choice between two alternatives is to be made by the provider, but a deeper analysis of this case is beyond the scope of this paper.

Mapping of Feature Names: An explicit mapping of feature names makes matching independent of coincidental synonyms or homonyms and allows to rename provider models in order to conform to the requestor’s alphabet. The construction of the mapping itself is not of interest here. We assume that alphabets are based on a shared ontology, for which mappings are provided.
I Did Not Order That: We prevent the loose interpretation of specifications, which would allow to add information where it does not contradict existing constraints. Instead we negate all features that belong to the alphabet but are not mentioned in the feature model, thus disabling their selection.

8. FUTURE WORK
This paper represents a first step towards the use of linear logic as a semantics for feature models in the context of service specification and matching. Based on the evaluation above, areas of future work include the following.

To provide a notation for feature diagrams distinguishing requestor from provider-side choices, Or and Alternative groups should appear in these two variants. Semantically, this should not affect the instances derivable from a model, but will have an effect on the matching: If the goal requires requestor-side choice (as is our default now), this must be matched by the offer of choice to the requestor. If the goal specifies provider-side choice, the offer could match this by dropping some of the alternatives.

To lift the operation of matching, here explained in terms linear logic only, to the visual notation of feature diagrams, we should devise a set of graphical transformation rules to derive a goal from a collection of offers. These would include, for example, rules to merge shareable features, drop optional provider features, remove alternatives, etc. The models should match if and only if a derivation exists relating the offers with the goal, serving as an illustration of the matching process. However, due to the inherent nondeterminism of rule-based systems, this approach would not yield an efficient decision procedure.

To address various algorithmic challenges, such as the derivation of all instances of a service feature model, the remainder of a goal with respect to an offer, and the operations of static or dynamic matching, mapping of service feature models into finite constraint satisfaction problems (CSP) should be considered. As opposed to approaches such as SAT and BDD, CSP allows integer constraints that can directly express multiplicities of features.

To align the approach with our own work [14] on partial (dynamic) matching of services based on visual contracts, the connection of service feature models with semantic service specifications will be investigated.

9. ACKNOWLEDGMENTS
We are grateful to José Luiz Fiadeiro for his comments and questions on an earlier draft of this paper. Muhammad Naeem’s PhD studies are supported by a grant from Higher Education Commission, Pakistan.

10. REFERENCES