Personalisation of Telecommunications Services as Combinatorial Optimisation

David Lesaint
BT, UK
david.lesaint@bt.com

Deepak Mehta
4C, UCC, Ireland
d.mehta@4c.ucc.ie

Barry O'Sullivan
4C, UCC, Ireland
b.osullivan@4c.ucc.ie

Luis Quesada
4C, UCC, Ireland
l.quesada@4c.ucc.ie

Nic Wilson
4C, UCC, Ireland
n.wilson@4c.ucc.ie

Abstract

Modern feature-rich telecommunications services offer significant opportunities to human users. To make these services more usable, facilitating personalisation is very important. Such personalisation enhances the users' experience considerably. The Session Initiation Protocol and Distributed Feature Composition architecture allow users to select and compose telecommunications network applications or features. In this paper we view feature composition as a configuration problem. We model feature composition using a variety of combinatorial optimisation paradigms. In particular, we present and evaluate an approach to finding optimal reconfigurations of network features when a user's preferences violate the technical constraints defined by a set of DFC rules.

Introduction

Information and communication services, from newsfeeds to internet telephony, are playing an increasing, and potentially disruptive, role in our lives. As a result, service providers seek personalisation solutions to allow customers control and enhance the way digital services are delivered; for example, call control has received much attention in Plain-Old Telephony Service and Intelligent Network environments. An outcome of that work is the emergence of *features* as fundamental primitives for personalisation (International Telecommunication Union 1993; 1997).

A feature is an increment of functionality that modifies the basic service behaviour, e.g., call-divert-on-busy, multimedia ring-back tone, interactive voice response, find-me, etc. Features are optional and must be activated to fulfill their role. Once activated, they execute automatically (e.g., call-logging) or interactively (e.g., call-transfer). Service personalisation refers to the problem of selecting which features should be active and when. In this context, a challenge is to provide effective tool support to service subscribers.

Any personalisation capability is intrisincally dependent on the way features are realised in the underlying service application architecture (Lesaint & Papamargaritis 2008). Modern architectures, notably those based on the Session Initiation Protocol (Rosenberg *et al.* 2002), are user-centric and delegate control over the selection and composition of applications. Still, the management of interactions between applications remains incomplete or restrictive. Distributed Feature Composition (Jackson & Zave 1998) provides a comprehensive methodology underpinned by a formal architecture model to address this issue.

Service personalisation in DFC is essentially a feature-based subscription configuration problem. We propose a combinatorial optimisation approach to the problem and explore how user preferences can be handled consistently with feature interaction resolution constraints. We introduce, and compare experimentally, constraint programming (CP), boolean satisfiability (SAT) and integer linear programming (ILP) techniques to compute optimal relaxations in the case of conflicting requirements. The results suggest the superiority of the CP approach.

Personalisation and Feature Interaction

Because features are meant to be intelligible and easy to operate, they are generally fine-grained and address specific concerns such as privacy, mobility, logging, etc. Consequently, providers organise their feature catalogues to ensure minimum redundancy and maximum combinability between features. Supporting feature composition is a source of interactions though. Informally, a feature interaction is "some way in which a feature modifies or influences the behaviour of another feature in generating the system's overall behaviour" (Bond et al. 2004).

Interactions are caused by resource contention (e.g., features sharing an audio channel), signal overloading (i.e., common signals being used for different purposes), the different ways features handle the same condition (e.g., call-divert and call-waiting on "busy"), etc. For instance, a do-not-disturb feature subscribed by a callee will block incoming calls and may cancel the effect of other features, e.g., a welcome announcement. Users may consider such interactions desirable or undesirable depending on the scenario.

Different solutions exist to detect, analyse and resolve interactions at design-time, subscription-time and/or runtime (Calder *et al.* 2003). Whatever the approach, these solutions tend to be architecture-specific since the very existence and form of interactions depend on the way features are modularised and composed at runtime. From a personalisation perspective, common concerns and requirements must be addressed irrespectively of architectures, notably:

- Roles. Features are designed to act on behalf of callers (e.g., originating-call-screening), callees (e.g, terminating-call-screening) or both (e.g., call-waiting). Users must then personalise their service consistently with this role classification.
- Multiple feature selection. Since features address separate concerns, users must have the flexibility to select multiple features, e.g., “screen incoming calls and welcome callers with a video clip”.
- Feature parameterisation. Some features need run-time access to user-specific operational data, e.g., a call-divert feature assumes a redirection address.
- Feature sequencing. Users may wish to specify a chronological ordering on the processing of features, e.g., “screen calls before diverting them”.
- Contextualisation. Different situations demand different service behaviours. Rather than having to continuously re-configure their service, users should have the flexibility to stipulate conditions on the activation of features. Conditions may refer to intrinsic or extrinsic session characteristics, e.g., “mute calls when I attend a seminar”.
- Preferences. Allowing users to compose features or express activation conditions can be a source of inconsistency if features conflict and/or contexts overlap, e.g., “call-divert at lunch time” and “voice-mail on Fridays”. If so, user preferences may be used to resolve inconsistencies.
- Priorities. Different individuals or entities may wish to control the way a service is configured, e.g., the end-user, his line manager, his department, etc. If so, priorities may be used to resolve conflicts.

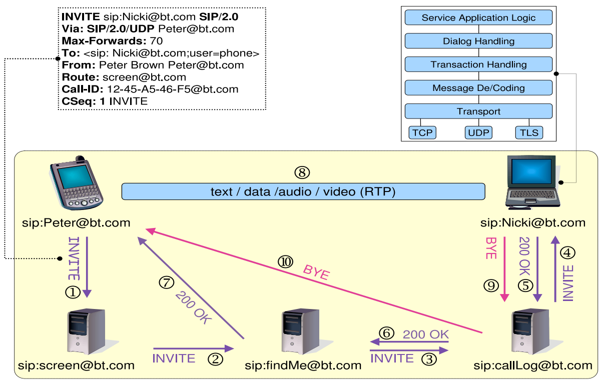
Below we review some personalisation solutions developed in the context of SIP and DFC to address these requirements.

SIP and DFC Architectures

SIP is an application layer signalling protocol used to establish, modify and terminate multimedia sessions. It underpins services as different as IP telephony, instant messaging, conferencing, presence awareness, IP-TV, video gaming or home appliance control (Sparks 2007). SIP is text-based and its messages can be transported with the Transmission Control Protocol (TCP/IP). It also relies on media protocols like the Real-Time Transport Protocol (RTP) to carry text, voice, video and data between endpoints.

The SIP specification prescribes an abstract architecture comprising entities such as user agents, proxy servers and registrars. Proxy servers route messages between user agents and dialogs are set up, modified, and terminated through a series of request-response transactions between adjacent entities on the connection path. Altogether, these mechanisms enable a dynamic chaining of entities during the dialog setup phase - a style of composition known as composition-by-proxying (see Figure 1).

Although the specification constrains the way entities manage the low-levels of the SIP stack (i.e., transport, message encoding/decoding, transaction and dialog management), it does not restrict their call control logic, e.g., the way they generate or proxy requests. Neither does it restrict the way they are implemented or deployed to network elements, nor does it impose any global routing policy. This makes SIP a very versatile protocol open to different service logic, implementation technologies and deployment models from peer-to-peer (Bryan & Lowekamp 2007) to centralised environments (Steinmann 2007).



```
INVITE sip:Nicki@bt.com SIP/2.0
Via: SIP/2.0/UDP Peter@bt.com
Max-Forwards: 70
To: <sip: Nicki@bt.com;user=phone>
From: Peter Brown Peter@bt.com
Route: screen@bt.com
Call-ID: 12-45-A5-46-F5@bt.com
CSeq: 1 INVITE
```

Figure 1: Message routing in a SIP architecture.

Personalising SIP services

Various capabilities have been proposed to program or exert control over SIP entities. SIP scripting languages, for instance, allow one to specify fine-grained call control rules in the form of scripts combining conditions and actions. Scripts are uploaded to devices or application servers and interpreted by embedded engines at runtime. Examples of this approach include CPL (Lennox, Wu, & Schulzrinne 2004) and LESS (Wu & Schulzrinne 2002). SIP APIs provide an alternative. The SIP servlet API, for instance, provides real-time and operational control over the invocations of SIP servlets (Java Community Process 2007).

The scope of these capabilities remains limited to the devices and servers where scripts, policies, rules or servlets are deployed. Other proposals like the Internet Multimedia Subsystem (IMS) (Poikselka *et al.* 2006) take a holistic approach and organise network architectures around SIP routers. The latter orchestrate the flow of messages between distributed applications by evaluating user-defined filter criteria. Although these solutions support some form of personalisation, none embrace a principled feature engineering approach to design applications and manage their interactions. The abstract DFC architecture which provides its own signalling and media protocols is relevant in this respect.

DFC

Similar to SIP and the IMS, DFC establishes a dialog between endpoints by routing a setup request encapsulating source and target addresses. Addresses may be changed along the way and DFC routers evolve the connection path accordingly. Starting from the box initiating the call, feature boxes are incorporated one after the other until a terminating box is reached. A router is used at each step to locate the next box and relay the setup request.

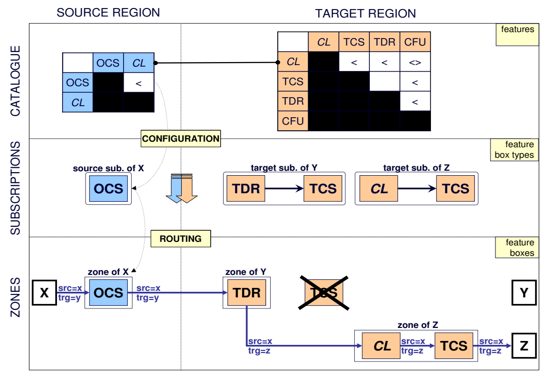


Figure 2: DFC: Catalogues, subscriptions and sessions.

The routing method, as a whole, consists of concatenating sequences of feature boxes called zones. The sequence of zones is itself split into a source region and a target region (see Figure 2). A source (resp., target) zone is bound to the source (resp., target) address of the request reaching its first box, that is, all its boxes act on behalf of the owner of the address under the assumption that he/she is the caller (resp., callee). For instance, the box Time-Dependent-Routing (TDR) in Figure 2 runs on behalf of subscriber Y in a callee role.

By default, the first source zone is associated with the source address of the initial setup request, e.g., zone of X in Figure 2. A change of source address in the source region, caused for instance by an identification feature, triggers the creation of a new source zone (Zave, Goguen, & Smith 2004). If no such change occurs in a source zone and the zone cannot be expanded further, routers switch to the target region. Likewise, a change of target address in the target region, as performed by TDR in Figure 2, triggers the creation of a new target zone. If no such change occurs in a target zone and the zone cannot be expanded further, the request is sent to the final box identified by the encapsulated target address.

DFC routers are only concerned with locating boxes and assembling zones into regions. They do not make decisions as to the type of feature boxes appearing in zones or their ordering. They simply fetch this information from the subscriptions preconfigured for each address in each region. When a zone is to be created for an address in a region, a router initialises a field of the setup request with the value of the associated subscription. It then locates a feature box corresponding to the first feature of the subscription and routes the request to the box. The routing process iterates by unfolding the above mentioned field until it is exhausted (as for Z in Figure 2) or the zone is interrupted (as for Y).

A DFC dialog may then be set up from multiple and partially unfolded subscriptions. Note that dialogs may also evolve into non-linear structures due to forking and joining features (e.g., call-conferencing, call-queue). Overall, this composition style allows subscribers to control dialog setups. It can also prevent undesirable interactions by making subscriptions, and thereby zones, "interaction-free".

To this end, DFC allows designers to formulate precedence constraints between features that every subscription must comply with. These constraints are uncovered by analysing all possible pairs of features in each region (Zave 2003). For each pair, this consists of determining which routing order, if any, could lead to an interaction. If so, a precedence constraint is created to impose the inverse order as for the case of Originating-Call-Screening (OCS) and Call-Logging (CL) in Figure 2. If no routing order is acceptable, two symmetric constraints are created to make the features mutually exclusive as for the case of CL and Call-Forwarding-Unconditional (CFU) in Figure 2.

This resolution method is safe, i.e., a constraint between two features will prevent their interaction irrespective of the zone the feature boxes might appear in, that is, whatever the session. The method is also complete relative to the constraints uncovered by designers and bearing in mind that its application scope is limited to zones. For instance, it does not tackle interactions that may arise between features subscribed by different addresses.

Configuring Subscriptions in DFC

The personalisation of a DFC service involves configuring a subscription from a feature catalogue. A catalogue is a set of source, target and reversible features - a subclass of features that are both source and target (like CL in Figure 2) and a set of precedence constraints between features in each region. A subscription is a subset of catalogue features and a set of user precedence constraints between features in each region. For instance, the subscription of Y in the target region includes the user precedence TDR$<$TCS.

Configuring a subscription involves selecting, parameterising and sequencing features in each region consistently with the catalogue constraints and other integrity rules (Jackson & Zave 2003). In particular, the source and target regions of a subscription must include the same reversible features in inverse order, i.e. source and target regions are not configured independently.

Checking the consistency of a subscription can be reduced to a cycle detection problem in a directed graph. Note first that each region of a catalogue may be modelled as a directed graph by mapping features to nodes and precedence constraints to arcs. Since the two regions of a catalogue have reversible features in common, their graphs may be merged into a single *catalogue graph* by inversing target constraints beforehand. The same transformation can be applied to subscriptions to generate *subscription graphs*. Proving the consistency of a subscription requires proving the acyclicity of the *extended graph* obtained by merging the subscription graph with the projection of the catalogue graph onto its node set. Since cycle detection is linear in time, this equivalence makes it possible to check subscriptions using interactive configuration systems (Lesaint *et al.* 2008). In this context, the following services may be provided to users submitting an input subscription: (**verification**) check the consistency of the subscription by checking the acyclity of the extended graph; (**partial completion**) if consistent, extend it with entailed precedence constraints by computing the partial order of the extended graph; (**filtering**) if consistent, compute its *anti-subscription*, i.e. the set of features and precedence constraints that would make it inconsistent if added; (**completion**) if consistent, suggest complete and consistent extensions by making its extended graph total and acyclic; (**revision**) if inconsistent, suggest relaxations by computing acyclic subgraphs of the extended graph. These tasks are formalised in the next section.

Formalisation

Let f_i and f_j be features, we write a precedence constraint of f_i before f_j as $\langle f_i, f_j\rangle$, or alternatively, p_{ij}. A *feature catalogue* is a tuple $\langle F, P\rangle$, where F is a set of features and P is a set of precedence constraints on F. Note that an exclusion constraint between features f_i and f_j expresses that these features cannot appear together in a feature subscription. We encode this as the pair of precedence constraints $\langle f_i, f_j\rangle$ and $\langle f_j, f_i\rangle$. The *transpose* of a catalogue $\langle F, P\rangle$ is $\langle F, P^T\rangle$ such that $\forall\langle f_i, f_j\rangle \in F^2 : \langle f_i, f_j\rangle \in P \Leftrightarrow \langle f_j, f_i\rangle \in P^T$. As described above, a source catalogue $\langle F_s, P_s\rangle$ and a target catalogue $\langle F_t, P_t\rangle$ can be composed into a single catalogue $\langle F_c, P_c\rangle \equiv \langle F_s \cup F_t, P_s \cup {P_t}^T\rangle$.

A *feature subscription* S of catalogue $\langle F_c, P_c\rangle$ is a tuple $\langle F, C, U, W_F, W_U\rangle$, where $F \subseteq F_c$, C is the projection of P_c on F, i.e., $P_c\downarrow_F = \{\langle f_i, f_j\rangle \in P_c : \{f_i, f_j\} \subseteq F\}$, U is a set of (user defined) precedence constraints on F, $W_F : F \rightarrow \mathbb{N}$ is a function that assigns weights to features and $W_U : U \rightarrow \mathbb{N}$ is a function that assigns weights to user precedence constraints. The value of S is defined by $\text{Value}(S) = \sum_{f\in F} W_F(f) + \sum_{p\in U} W_U(p)$. Note that a weight associated with a feature signifies its importance for the user. These weights can be elicited using data-mining or analysis of user interactions.

A feature subscription $S = \langle F, C, U, W_F, W_U\rangle$ is defined to be *consistent* if and only if the corresponding graph $\langle F, C \cup U\rangle$ is acyclic. Due to the composition of the source and target catalogues into a single catalogue, a feature subscription S is consistent if and only if both source and target regions are consistent in the DFC sense. Determining whether a feature subscription $\langle F, C, U, W_F, W_U\rangle$ is consistent or not can be checked in $\mathcal{O}(|F| + |C| + |U|)$ by using Topological Sort (Cormen, Leiserson, & Rivest 1990).

Let $S = \langle F, C, U, W_F, W_U\rangle$ be an inconsistent feature subscription. The set R_S of all relaxations of S, is the set of consistent feature subscriptions $\langle F_i, C_i, U_i, W_{F_i}, W_{U_i}\rangle$ such that $F_i \subseteq F$, $C_i = P_c\downarrow_{F_i}$, $U_i \subseteq U\downarrow_{F_i}$, $W_{F_i} = W_F\downarrow_{F_i}$, and $W_{U_i} = W_U\downarrow_{U_i}$. We say that $S_i \in R_S$ is an *optimal relaxation* of S if it has maximum value among all relaxations, i.e., if and only if there does not exist $S_j \in R_S$ such that $\text{Value}(S_j) > \text{Value}(S_i)$. Finding an optimal relaxation of an inconsistent feature subscription is NP-Hard.

Given a catalogue $\langle F_c, P_c\rangle$ and a consistent feature subscription $S = \langle F, C, U, W_F, W_U\rangle$, the *anti-subscription* is the tuple $\langle F_a, P_a\rangle$ defined as follows. $f \in F_c$ is an element of F_a if and only if the directed graph associated with the subscription obtained after adding feature f, i.e., $\langle F \cup \{f\}, P_c\downarrow_{F\cup\{f\}} \cup U\rangle$ is cyclic; pair $\langle f_i, f_j\rangle \in F^2$ is in P_a if and only if the directed graph associated with the subscription obtained after adding precedence $\langle f_i, f_j\rangle$, i.e., $\langle F \cup \{f_i, f_j\}, P_c\downarrow_{F\cup\{f_i,f_j\}} \cup U \cup \{\langle f_i, f_j\rangle\}\rangle$ is cyclic.

The definition of anti-subscription suggests one way of implementing this task. To test whether a feature/precedence belongs to the anti-subscription we check the consistency of the resulting subscription. As there are $\mathcal{O}(|F_c|^2)$ possible checks, the overall complexity of computing an anti-subscription is $\mathcal{O}(|F_c|^2 \times (|F| + |C| + |U|))$.

Given a consistent feature subscription $S = \langle F, C, U, W_F, W_U\rangle$, the *partial order* of the subscription is the transitive closure $(C \cup U)^*$ of the relation $C \cup U$. The worst-case complexity of finding this transitive closure is $\mathcal{O}(|F|^3)$. A *total order* of consistent subscription S is a topological sort of the directed graph $\langle F, C \cup U\rangle$, i.e., a total order extending the relation $C \cup U$. The worst-case complexity of finding such a total order is linear in time with respect to the size of the corresponding graph.

Note that checking the consistency of a feature subscription, and computing the anti-subscription, the partial order and a total order are all polynomial tasks. The most challenging task is finding an optimal relaxation of a subscription when it is not consistent, since it is NP-Hard. In the remainder of the paper we focus only on this particular task.

Finding an Optimal Relaxation

We have used a constraint programming (CP) (Rossi, van Beek, & Walsh 2006), a Partial Weighted Maximum Boolean Satisfiability (PWMSAT) (Argelich & Manyà 2007) and an Integer Linear Programming (ILP) (Vanderbei 2007) technique to solve this particular task. Because of space restrictions, we shall only present the constraint programming formulation for finding an optimal relaxation. We empirically evaluated all our formulations of the different approaches. A constraint programming formulation for finding an optimal relaxation of the input subscription $\langle F, C, U, W_F, W_U\rangle$, when inconsistent, is outlined below.

Variables. We associate each feature $f_i \in F$ with two variables: a *Boolean variable* bf_i and an *integer variable* pf_i. The variable bf_i is instantiated to 1 or 0 depending on whether f_i is included in the computed subscription or not, respectively. The domain of each integer variable pf_i is $\{1, \ldots, |F|\}$. We associate each user precedence constraint $p_{ij} \equiv \langle f_i, f_j\rangle \in U$ with a *Boolean variable* bp_{ij}. The variable bp_{ij} is instantiated to 1 or 0 depending on whether p_{ij} is respected in the computed subscription or not, respectively.

Constraints. A catalogue precedence constraint $p_{ij} \in C$ that feature f_i should be before feature f_j can be expressed as $bf_i \wedge bf_j \Rightarrow (pf_i < pf_j)$. A user precedence constraint $p_{ij} \in U$ that f_i should be placed before f_j in their subscription can be expressed as $bp_{ij} \Leftrightarrow (bf_i \wedge bf_j \wedge (pf_i < pf_j))$. Note that if a user precedence constraint holds then the features f_i and f_j are included in the subscription and also the feature f_i is placed before f_j, that is, the selection variables bf_i and bf_j are instantiated to 1 and $pf_i < pf_j$ is true.

The objective function for finding an optimal relaxation of the input feature subscription can be expressed as follows:

$$\text{Maximize} \sum_{f_i\in F} bf_i \times W_F(f_i) + \sum_{p_{ij}\in P} bp_{ij} \times W_U(p_{ij}).$$

A depth-first branch and bound algorithm is used to find an optimal relaxation. Although the worst-case time complexity of the algorithm is exponential, its efficiency can be improved significantly by computing tight upper bounds at each node of the search tree. The quality of the upper bound can be improved by increasing the level of local consistency that is maintained at each node of the search tree. The different levels of local consistency we have used within branch and bound search are *Generalized Arc Consistency* (GAC) (Bessière & Régin 1997) and a form of *mixed consistency* (Dooms, Deville, & Dupont 2005), which means enforcing different levels of consistency on different variables.

Evaluation of Different Approaches

We generated and experimented with a variety of *random catalogues* and many classes of *random feature subscription* (RFS). A uniform distribution was used for randomly selecting elements from the sets. A random catalogue is defined by a tuple $\langle f_c, b_c, T_c\rangle$. Here, f_c is the number of features, b_c is the number of binary constraints and $T_c \subseteq \{<, >, <>\}$ is a set of types of constraints. Note that $f_i <> f_j$ means that in any given subscription both f_i and f_j cannot exist together. A random catalogue is generated by selecting b_c pairs of features randomly from the set of all $f_c(f_c - 1)/2$ pairs of features. Each selected pair of features is then associated with a type of constraint that is selected randomly from T_c. A RFS is defined by a tuple $\langle f_u, p_u, w\rangle$. Here, f_u is the number of features that are selected randomly from f_c features, p_u is the number of user precedence constraints between the pairs of features that are selected randomly from $f_u(f_u - 1)/2$ pairs of features, and w is an integer greater than 0. Each feature and each user precedence constraint is associated with an integer weight that is selected randomly between 1 and w inclusive.

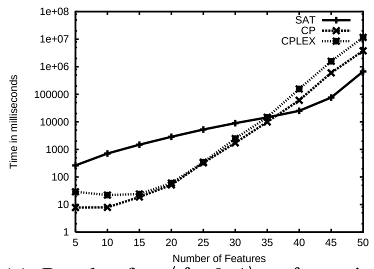
We generated catalogues of the following forms: $\langle 50, 250, \{<, >\}\rangle$, $\langle 50, 500, \{<, >, <>\}\rangle$ and $\langle 50, 750, \{<, >\}\rangle$, but due to space limitations, we present the details only for the last class, since they are representative of all others. For each random catalogue, we generated classes of RFSs of the following forms: $\langle 10, 5, 4\rangle$, $\langle 15, 20, 4\rangle$, $\langle 20, 10, 4\rangle$, $\langle 25, 40, 4\rangle$, $\langle 30, 20, 4\rangle$, $\langle 35, 35, 4\rangle$, $\langle 40, 40, 4\rangle$, $\langle 45, 90, 4\rangle$ and $\langle 50, 5, 4\rangle$. For each class 10 instances were generated and their mean results are reported in this paper.

The CP model for finding an optimal relaxation was implemented and solved using Choco (`http://choco-solver.net`), a Java library for constraint programming systems. In our implementation we maintain restricted singleton generalized arc consistency (Prosser, Stergiou, & Walsh 2000) on Boolean variables and GAC on the remaining variables. Our experience is that maintaining (restricted) singleton generalized arc consistency on the Boolean variables often reduces the search space and time of the branch and bound algorithm significantly. The PWMSAT model of the problem was implemented and solved using SAT4J (`http://www.sat4j.org`), an efficient library of SAT solvers in Java. The ILP model of the problem was solved using ILOG CPLEX (`http://www.ilog.com/products/cplex/`). All the experiments were performed on a PC Pentium 4 (CPU 1.8 GHz and 768MB of RAM) processor. The performance of all the approaches was measured in terms of search nodes (#nodes) and runtime in milliseconds (time). We used the time limit of 4 hours to cut the search. In the tables, a number in parenthesis after the time marks the number of instances not solved (last two lines).

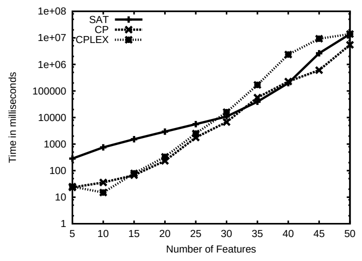
Table 1: A sample of results on catalogue $\langle 50, 750, \{<, >\}\rangle$.

	PWMSAT		CPLEX		CP	
$\langle f, p\rangle$	#nodes	time	#nodes	time	#nodes	time
$\langle 10, 5\rangle$	226	761	22	122	14	40
$\langle 15, 20\rangle$	865	1,614	273	976	36	132
$\langle 20, 10\rangle$	1,886	3,016	877	2,990	67	301
$\langle 25, 40\rangle$	7,247	6,861	12218	35,216	236	2,732
$\langle 30, 20\rangle$	10,726	11,042	28,503	107,988	669	6,096
$\langle 35, 35\rangle$	24,945	26,161	159,930	1,582,155	1,688	21,899
$\langle 40, 40\rangle$	57,429	84,291	400,546	5,199,400	4,883	75,031
$\langle 45, 90\rangle$	532,684	2,571,336 (1)	823,073	14,400,025 (10)	13,397	336,076
$\langle 50, 4\rangle$	182,869	472,325	808,054	14,400,025 (10)	44,390	536,975

We present a sample of our detailed results in Table 1. These results suggest that our CP approach performs best overall, solving all instances. Even though in very few cases it is outperformed by the other two approaches, it manages to make a significant gap in all other cases. Experimental results suggest that our ILP and PWMSAT approaches require noticeably more time than the CP approach.



(a) Results for $\langle f_u, 0, 1\rangle$: f_u varies from 5 to 50 in steps of 5.



(b) Results for $\langle f_u, p_u, 1\rangle$: $f_u = p_u$ and f_u varies from 5 to 50 in steps of 5.

Figure 3: Comparing the different approaches.

Figure 3(a) presents a comparison of the different approaches in terms of their runtimes for subscriptions, when $|U| = 0$ and the weight of each feature is 1. The runtimes of the approaches for the classes of RFS when $|F| = |U|$ are presented in Figure 3(b). The PWMSAT approach is outperformed by the other two approaches when $|F|$ is less than 35 and 25 in Figures 3(a) and 3(b), respectively. However, it performs better than the other approaches for $|F| > 35$ and $|U| = 0$. It would be interesting to find out whether the PWMSAT approach deteriorates when $|F| > 50$. We observe that when $|F| > 35$, the runtime of CPLEX is the worst. In Figure 3(b), when $|F| = 50$, neither the ILP approach nor the PWMSAT approach managed to solve all the instances. In fact, this is why their average runtime, for the case of 50 features, is basically the timeout. The gap between the CP approach and the other approaches, for the case of 50 features in Figure 3(b), does not seem very remarkable because the timeout is relatively close to the time spent by the CP approach. However, other experiments suggest that the gap would be more significant if the timeout were higher.

Conclusions

We have presented an approach to personalised feature composition in telecommunications systems. Our architecture is based on SIP and DFC that provide a sound foundation for developing safe and flexible personalisation solutions. We considered feature composition as a configuration problem. We presented, and evaluated, an optimisation-based approach to finding optimal reconfigurations of network features when the user's preferences violate the technical constraints defined by a set of DFC rules. Our results suggest that finding an optimal relaxation for concrete catalogues (e.g., (Bond *et al.* 2005) presents a catalogue of 25 features) is feasible using constraint programming. The results also show that CP outperforms the other approaches. Another benefit of using CP is that more complex constraints on features can be formulated in a more natural way. In our future work we will develop approaches to managing context-rich subscriptions.

Acknowledgements

This material is based upon research supported by Science Foundation Ireland (Grant No. 05/IN/I886), and the Embark Initiative (Fellowship Nos. CT1080049908 and CT1080049909). The authors would also like to thank Hadrien Cambazard for his support in using Choco.

References

Argelich, J., and Manyà, F. 2007. Partial Max-SAT solvers with clause learning. In *SAT*, 28–40.

Bessière, C., and Régin, J.-C. 1997. Arc consistency for general constraint networks: Preliminary results. In *IJCAI*, 398–404.

Bond, G. W.; Cheung, E.; Goguen, H.; Hanson, K. J.; Henderson, D.; Karam, G. M.; Purdy, K. H.; Smith, T. M.; and Zave, P. 2005. Experience with Component-Based Development of a Telecommunication Service. In *CBSE 2005*, 298–305.

Bryan, D. A., and Lowekamp, B. B. 2007. Decentralizing SIP. *ACM Queue* 5(2):34–41.

Calder, M.; Kolberg, M.; Magill, E. H.; and Reiff-Marganiec, S. 2003. Feature Interaction: a critical review and considered forecast. *Comp. Networks* 41(1):115–141.

Cormen, T.; Leiserson, C.; and Rivest, R. 1990. *Introduction to Algorithms*. The MIT Press.

Dooms, G.; Deville, Y.; and Dupont, P. 2005. CP(Graph): Introducing a graph computation domain in constraint programming. In *CP 2005*.

International Telecommunication Union. 1993. Introduction to Intelligent Network Capability Set 1. Recommendation Q.1211, ITU, Geneva, Switzerland.

International Telecommunication Union. 1997. Introduction to Intelligent Network Capability Set 2. Recommendation Q.1221, ITU, Geneva, Switzerland.

Jackson, M., and Zave, P. 1998. Distributed Feature Composition: a Virtual Architecture for Telecommunications Services. *IEEE TSE* 24(10):831–847.

Jackson, M., and Zave, P. 2003. *The DFC Manual*. AT&T.

Java Community Process. 2007. SIP servlet version 1.1. Java Specification Request 289, JCP.

Lennox, J.; Wu, X.; and Schulzrinne, H. 2004. Call Processing Language (CPL): A Language for User Control of Internet Telephony Services. RFC 3880 (standard), IETF.

Lesaint, D., and Papamargaritis, G. 2008. Personalised Communications. In *Service Chain Management - Technology Innovation for the Service Business*. 187–203.

Lesaint, D.; Mehta, D.; O'Sullivan, B.; Quesada, L.; and Wilson, N. 2008. A Constraint-Based System for the Configuration of Subscriptions to Feature-Based Telecommunications Services. Patent report, BT, Ipswich, UK.

Poikselka, M.; Mayer, G.; Khartabil, H.; and Niemi, A. 2006. *The IMS: IP Multimedia Concepts and Services*. John Wiley & Sons.

Prosser, P.; Stergiou, K.; and Walsh, T. 2000. Singleton Consistencies. In Dechter, R., ed., *CP-2000*, 353–368.

Rosenberg, J.; Schulzrinne, H.; Camarillo, G.; Johnston, A. B.; Peterson, J.; Sparks, R.; Handley, M.; and Schooler, E. M. 2002. SIP: Session Initiation Protocol. RFC 3261 (standard), IETF.

Rossi, F.; van Beek, P.; and Walsh, T. 2006. *Handbook of Constraint Programming*. Elsevier Science Inc.

Sparks, R. 2007. SIP: Basics and Beyond. *ACM Queue* 5(2):22–33.

Steinmann, M. J. 2007. Unified Communications with SIP. *ACM Queue* 5(2):50–55.

Vanderbei, R. J. 2007. *Linear Programming: Foundations and Extensions*. Springer.

Wu, X., and Schulzrinne, H. 2002. Programmable End System Services Using SIP. In *2nd New York Metro Area Networking Workshop*.

Zave, P.; Goguen, H.; and Smith, T. M. 2004. Component Coordination: a Telecommunication Case Study. *Computer Networks* 45(5):645–664.

Zave, P. 2003. An Experiment in Feature Engineering. In McIver, A., and Morgan, C., eds., *Programming Methodology*. Springer-Verlag. 353–377.