# IP routing to serve targeted tracing in software product lines

# Le routage IP au service de la traçabilité dans les lignes de produits logiciels

**Zineb Mcharfi**

*SIME Laboratory, ENSIAS, Mohammed V University, Rabat, Maroc*

*zineb.mcharfi@gmail.com*

**Bouchra El Asri**

*SIME Laboratory, ENSIAS, Mohammed V University, Rabat, Maroc*

*elasri@ensias.ma*

**Abdelaziz Kriouile**

*SIME Laboratory, ENSIAS, Mohammed V University, Rabat, Maroc*

*kriouile@ensias.ma*

**Résumé**

Tracer est un élément clé pour garantir une maintenance efficace et une facilité d'utilisation du système. Ceci dit, lors de la mise en place d'un système ou d'une application, adopter une solution de traçabilité n'est généralement pas une priorité du fait qu'elle est considérée comme source d'effort et de coûts additionnels. Dans le cas des Lignes de Produits Logiciels, la traçabilité ajouterait de la complexité au système. Cependant, à travers nos recherches sur la traçabilité dans les Lignes de Produits Logiciels, nous avons constaté qu'une traçabilité ciblée était rentable en termes de coûts et de Retour sur Investissement (ROI). Par conséquent, nous proposons dans le présent article, en continuité de nos précédentes publications, un algorithme détaillé permettant d'implémenter une traçabilité ciblée dans les Lignes de Produits Logiciels, inspiré des protocoles de routage dans les réseaux IP.

**Abstract**

Tracing is a key element to guarantee efficient maintenance and ease to use systems, especially for complex large scale ones like Software Product Lines. However, while developing a system or an application, tracing is not always considered as a priority, as it generates additional effort and cost. For Software Product Lines, generating trace links can even add complexity to the system. However, as part of our research on tracing in Software Product Lines, we found that adopting a targeted tracing approach is more profitable for costs and Return On Investment (ROI). Therefore, we propose in this paper, as continuity to previous publications, a detailed algorithm to implement targeted traceability links in a Software Product Line, inspired from IP routing protocols.

**Mots-clés**

Lignes de Produits Logiciels, traçabilité ciblée, liens de traçabilité, réseau, routage

**Keywords**

Software Product Lines, targeted traceability, trace links, network, routing

# 1. Introduction

The importance of traceability in software engineering has been highlighted since 1968 (Cleland-Huang, Gotel *et al.*, 2012). However, traceability is still rarely adopted in practice, as it is laborious, usually manual, time and resource consuming, and error prone (Ramesh et Jarke, 2001). Software Product Lines (SPLs) are also concerned. This motivated our research work to setup a SPL tracing approach that can help improve the quality of a SPL solution without adding complexity to this large scale system.

In some previous publications, we presented our work on tracing Return On Investment in SPLs (Mcharfi, El Asri *et al.*, 2015a), (Mcharfi, El Asri *et al.*, 2015b) and proposed a model, MeTraSPL to study the cost of tracing in SPLs. Based on our study results, we worked on an algorithm that would allow efficient tracing in SPLs, without unnecessary costs. We introduced this algorithm in a previous publication (Mcharfi,, El Asri *et al.*, 2016), and we are presenting the algorithm generalization as well as a detailed description in the present article.

The remaining of this paper is as follows: in section 2 we introduce Software Product Lines and traceability in those systems. We discuss IP networks routing protocols in section 3, and describe the algorithm we propose based on dynamic routing in IP networks. In section 4 we present a case study for tracing in a telecom operator SPL and how to apply our algorithm, before concluding in section 5.

# 2. Motivations and Background

Software Product Lines are promising systems that help save time and resources while producing in large quantities. However, to reach those goals, Product Lines have a complex composition, based on variability. Therefore, maintaining such systems can be hard task if the process is not systematic. An element that can help maintaining Software Product Lines is tracing.

In this paragraph, we will discuss traceability in Software Product Lines and how it can be optimized in order not to add complexity to the system.

## 2.1 Software Product Lines

Software Product Lines are “a set of software-intensive systems that share a common, managed feature set satisfying a particular market segment’s specific needs or mission and that are developed from a common set of core assets in a prescribed way” (Northrop, 2002). SPLs help producing numerous products starting from common components called “core assets” (e.g., architecture, requirements, test plans, schedules, budgets and processes description). In addition to those common elements, products can have their specific components, which can help producing specific and personalized products. SPLs are based on organized reusing approach.

SPLs represent an up-front, proactive reuse demarche (Krueger, 2002): they are based on a production plan, involve both technical and organizational management, are direct results of the organization strategy, and are used to reach predictable results.

A Software Product Line is a combination of three major interacting elements (Northrop, 2002), (Clements and Northrop, 2005): (i) core asset development, also called Domain Engineering (DE), (ii) product development, called Activities Engineering (AE) and (iii) the technical and organizational management that orchestrates those two activities.

It is clear then that Software Product Lines are complex systems, composed of many elements to take into consideration while adopting a tracing approach.

## 2.2 Tracing in Software Product Lines

Software traceability is the ability to connect two different artefacts throughout their lifetime to help understand and describe the software, as well as take decisions during development and maintenance phases. It deals with components and relationship between them (Cleland-Huang, Gotel *et al.*, 2012), producing and maintaining consistent documentation, verifying the completeness of requirements implementation (Cleland-Huang, Gotel *et al.*, 2014), and being independent from individual knowledge (Lindvall and Sandahl, 1996). Traceability improves system quality. It also helps choosing architectures and identifying errors, and facilitates communication between stakeholders (Anquetil, Kulesza *et al.*, 2010). It is very helpful for maintenance and evolution as it allows analyzing and controlling the impact of changes (Cavalcanti, do Carmo Machado *et al.*, 2011), which is very useful in SPLs context as traces help identifying elements impacted by changes in the Product Line.

Traceability in SPL can be used either while developing, for short term purposes (e.g., to verify and validate requirements implementation), or in maintenance phase, for long term use (e.g., artifacts understanding, change management and components reuse) (Cleland-Huang, Gotel *et al.*, 2012), (Ramesh and Jarke, 2001), (Spanoudakis and Zisman, 2005). However, many difficulties can be faced when implementing traceability in SPL (Jirapanthong and Zisman, 2005): (i) larger documentation than for traditional software development; (ii) documents heterogeneity; (iii) need to link between different products and between them and the SPL architecture. Also, unlike software engineering approaches for single systems, SPL introduces a complex dimension: variability. Variability represents an additional difficulty for traceability in SPL as one needs to understand its consequences during the development phases (Jirapanthong and Zisman, 2005).

Therefore, we worked on efficient tracing in SPLs as described in next paragraphs.

## 2.3 Efficient Tracing in Software Product Lines

The main subject of our researches is tracing in SPLs and how to optimize cost and effort while implementing a traceability approach in a SPL. We tried to define how the cost of implementing a SPL responds to a specific tracing method, what are the elements that impact that cost and how do they impact it. To answer those questions, we proposed in previous publication (Mcharfi, El Asri *et al.*, 2016) a model called MeTraSPL, based on the well-known effort measuring approach COCOMO II.

Based on our MeTraSPL model, we can state that, as shown in figures Fig. 1 and Fig. 2 bellow, tracing in SPLs is profitable on the mid (Application Engineering phase) and long terms (maintenance phase), while using targeted tracing approaches.

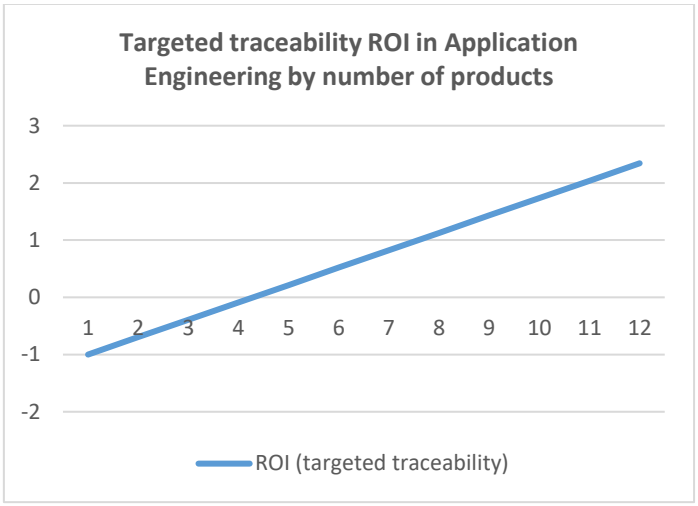


Fig. 1: Targeted traceability ROI in Application Engineering by number of products (Mcharfi, El Asri *et al.*, 2015b)

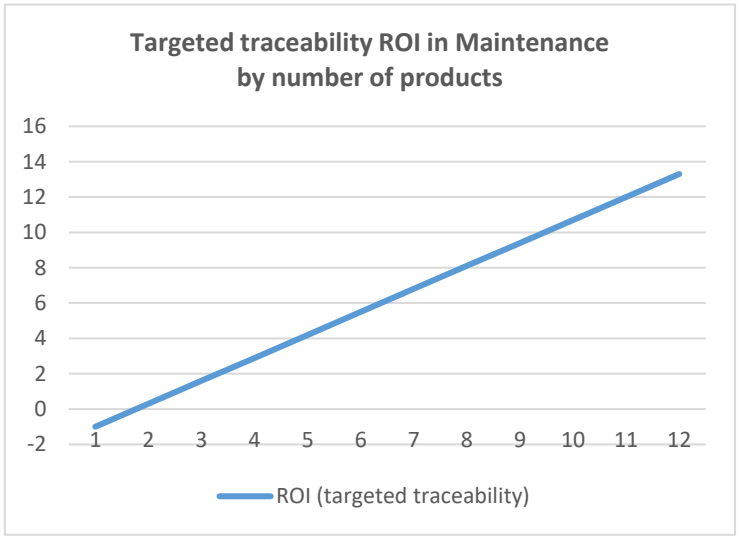


Fig. 2 Targeted traceability ROI in Maintenance by number of products (Mcharfi, El Asri *et al.*, 2015b)

Based on the previous findings, we propose in the next paragraph an algorithm for targeted tracing in Software Product Lines.

# 3. Outlook and contribution

Throughout our research work, we tried to define a methodology to optimize traceability in Software Product Lines, including the links between the features and the architecture model.

Therefore, we worked on an analogy that we established between traceability in SPL and dynamic routing in IP network.

In the next paragraphs, we will define some key elements related to routing in IP networks, in order to understand the analogy we established between tracing in Software Product Lines and dynamic routing in IP networks.

## 3.1 Routing in IP Networks

According to Cisco definition, "routing is the act of moving information across an internetwork from a source to a destination" (Cisco, 2010). Routing is about exchanging information between two elements that belong to different network, throughout some "intermediated" element, named routers (Medhi and Ramasamy, 2018).

To reach that goal and be able to link between two network components A and B, two major support elements are needed: addresses and routing tables. The address informs mainly about the element's corresponding network or subnetwork. A routing table uses addresses to help router identify the path where to send the information. It contains information about the next router for each of the related networks (Medhi and Ramasamy, 2018). Any change in the network configuration must be reported in the routing table of each router belonging to that network.

A routing table can be static, manually updated by network administrator (this is the case for simple, rarely changing networks) or dynamic, updated by some routing protocol. In the context of our work, we are interested in those dynamic routing protocols as we are working on complex systems such as Software Product Lines, and maintaining manually a huge routing table (there is an important number of components in a SPL, and connections are complex) is heavy and error prone.

### Dynamic Routing Protocols

A dynamic routing protocol is the set of actions by which a router knows the network status and how to reach, in any situation, his neighbours, using his routing table.

There are two main families of dynamic routing protocols:

**Interior gateway protocols**: protocols for exchanging data between elements of the same autonomous system (subnetwork). Their role is also to allow the restoration of links in case of failure (Bonaventure, 2010). Depending on the routing algorithm used, an interior gateway protocol can be one of two types: distance-vector routing protocol or link-state routing protocol.

- Distance-vector routing protocols: In these protocols, the router has no visibility over the entire network. It begins with a routing table that has only one entry, the one related to the router itself. Then the router sends this information to neighboring routers, and in return receives the entries from their routing tables. The router then updates its own table based on the received information. The distance vector information exchanged between routers contains the entries of the routing tables with a metric that represents the distance separating the router from this entry. This distance is measured in number of hops. It represents the number of intermediate routers that the information should go through before reaching the final destination. This is called the cost of the road (Bonaventure, 2010).
- Link-state routing protocols: For those protocols, data is routed using the shortest path, based on Dijkstra algorithm. Unlike distance-vector routing protocols, a router using link-state protocol must know the whole network topology. This is done by exchanging messages between routers each time network topography is modified. One of the most popular protocols of this category is the Open Shortest Path First (OSPF) protocol.

**Exterior gateway protocols**: These are protocols that allow communication between elements of two different autonomous systems. Knowing that each of them uses its own internal routing protocol, the external routing protocol allows them to communicate in a "common language" understood by all. Right now, the only protocol used for exterior routing in IP network is the Border Gateway Protocol (BGP).

## 3.2 Analogy between Tracing in Software Product Lines and Routing in IP Networks

### 3.2.1 Components Analogy

In Product Lines, the links between features and architectural components are many-to-many: a feature can be implemented by one or more components, and a component can implement one or more features (Fig. 3). Regarding the links from features to components in a SPL, the architecture is similar to IP network architecture. We divide the SPL network into many subnetworks, equivalent to autonomous systems in Internet networks. A component can belong to one or more systems.

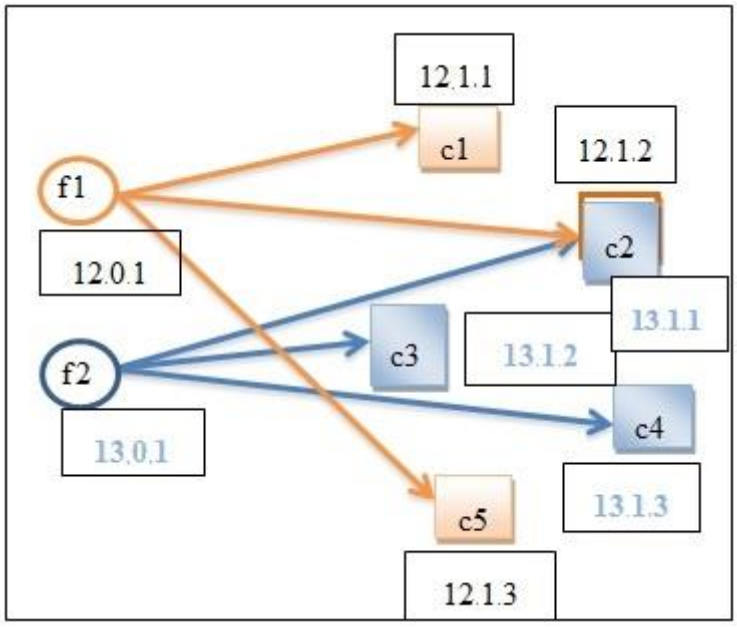


Fig. 3 Domain and component addresses in a Software Product Line

Next step is addresses assignment. This is a key element in our tracing approach. Each element of our SPL network is identified by an address. Components that may belong to different subnetworks have a different address for each subnetwork (e.g. in figure 3, component c2 is part of the two subnetworks 12 and 13, and therefore have two addresses: 13.1.1 and 12.1.2).

Also, as for IP networks, and depending on network objectives, some components have their own “routing table” to store information about their neighbours.

Another important element in components analogy between IP networks and SPL tracing is the communication protocol. In fact, as explained earlier, there are two types of IP dynamic routing protocols: Interior and exterior gateway protocols. This is due to internet diversity and heterogeneity. In that situation, a universal protocol is needed so as the components can

communicate with each other. In the algorithm we are proposing for SPLs, we consider that the environment is homogeneous and all components speak the same language.

### 3.2.2 Protocols Analogy

As described in a previous paragraph, routing in IP networks can be static or dynamic. For the traceability algorithm we are proposing in SPLs, and because of the complexity of these large-scale systems, we adopt an approach that is similar to dynamic routing in IP networks. In fact, Software Product Lines are complex and large-scale systems, and manual traceability would be expensive in terms of time and resources, and it is error prone.

Also, as illustrated through our previous work (Mcharfi, El Asri, *et al*. 2015b), tracing Return on Investment in the mid and long term is more interesting in the case of targeted tracing approach. Therefore, we propose a tracing algorithm inspired from distance-vector routing protocols as in that case, each element needs to know only its neighbours, not all the network topology like for Link-state routing protocols. Adopting a similar approach for SPL's traceability would make it possible to target the links to be traced in the product line. In fact, many changes can occur in the mid and long term of a Product Line life cycle: new artefacts added, other deleted or updated... In addition, in Application Engineering phase, different elements might be used for every specific product. Maintaining a global vision of the whole SPL "network", as it is the case for Link-state routing protocols, will cost.

## 3.3 Our Targeted Tracing Algorithm for Software Product Lines

The first step in our algorithm is to assign addresses to items in our Product Line. We tried to group together in the proposed addressing system the maximum amount of information needed. Each address is therefore composed of three parts: the domain identifier, the type of the element (feature = 0, component = 1) and the identifier of the element (Fig. 3). These elements are grouped into domains. We identify a domain by a feature with the components to implement it. Therefore, each feature defines one and only one domain, and a component can be part of one or more domains and will therefore have one or more addresses.

Also, with the objective of cost optimization with targeted traceability, we identify levels of granularity of traceability. Depending on the degree of granularity desired, the links between the elements of our Product Line will be more or less constructed in the Domain Engineering phase. The other links can be reconstituted if necessary thanks to our algorithm. In what follows, we consider for our algorithm a granularity of level 1: we trace to the nearest components of the feature.

We choose to trace only the common artefacts, the others are specific to the product to be generated, and therefore are used in the Applications Engineering phase.

For the communication and routing protocol, as mentioned earlier, we choose the Vector-distance protocol as it is flexible compared to link-state protocol, and helps not adding complexity to the solution as we are working on large-scale system.

Another important point: trace links maintenance. Indeed, one of the restrictive point elements in a traceability process, and which could deter its implementation, is the maintenance of traces links. In product lines, adding, deleting, or updating a component impacts not only one product but several. In addition, several elements, therefore links, are part of the composition of the product line, and maintaining all these links (even in the context of a targeted traceability) can be a heavy operation. For this, in the routing algorithm we propose by analogy to IP networks, the links needed for the traceability of the platform are updated dynamically, no specific maintenance routine is necessary.

In the algorithm we are proposing, we take into consideration the following points:

- Only feature-component and component-component links are affected
- We assign each link a status. All the links have a status = 1 when created.
- We use the pair (@, status) to distribute the information through the trace links.

The algorithm we are proposing helps:

- Set up a vertical and targeted traceability process
- Trace from a feature F to a component C and vice versa
- Detect any changes in the product line platform architecture (example: add, remove or modify an item)
- Update the traceability links dynamically

Following is the description of the algorithm we are proposing.

D is a domain with cardinality card(D) and identifier id(D). D is defined by the feature f($@_f$). $@_f$ is the address of f such as:

- $@_f$.DomaineIdentifier = Id(D)
- $@_f$.type = 0
- $@_f$.identifier = 1

c($@_c$) is a component in the domain D of the Product Line. $@_c$ is the address of c.

- $@_c$.DomaineIdentifier = Id(D)
- $@_c$.type = 1
- $@_c$.identifier = card(D)+1 (each time a new element is added to the Domain D, the cardinality of the latter is incremented by 1).

Let us consider different scenarios of changes in a SPL:

**If changes in the feature f**: Changes in f (delete or update) impact the components that implement it and should, therefore, be notified. These components are none other than those belonging to the same domain D as the feature f. They are identified thanks to the field Id(D) of their address. As for routing protocols, change information is broadcasted as follows:

- If it is a removal, $status_f$ is set to 0 to mention it is a removal. Before being removed, feature f send the pair ($@_f$, $status_f$) to first level neighbour components so as they can update their routing tables.
- If it is an update, as for removal, the pair ($@_f$, $status_f$) is broadcasted to first level neighbour components, with $status_f$ = -1 to say it's an update.

If changes on f impact a component c that belong to two domains D and D', the feature f' from D' might be impacted. In that case, component c transfers the new value ($@_f$, $status_f$) to feature f', identified using component c routing table.

As described earlier, in order to optimize tracing costs, we identify levels of granularity depending on tracing objectives. In our case, it is set to 1: we only want to trace first level components. If set to 2, the new value of pair ($@_f$, $status_f$) must be sent to components from second level, sons of first level components, and so on…

**If changes in component c**: The actions to be done depend on the number of addresses the component has:

- If c has one address, the new value of the pair ($@_c$, $status_c$) ($status_c$ = 0 if c removed, $status_c$ = -1 if updated) is sent to the feature f and, depending on granularity degree, to the other components. If f is impacted, then the feature change algorithm is to be unrolled.
- If c has many addresses, the new value of the pair ($@_c$, $status_c$) ($status_c$ = 0 if c removed, $status_c$ = -1 if updated) is sent to the features f and f' and, depending on granularity degree, to the other components in D and D'. If f and/or f' is impacted, then the feature change algorithm is to be unrolled for f and/or f'.

In the next paragraph, we apply this tracing algorithm in the context of a Software Product Line solution for managing telecommunication operator offers.

# 4. Case study: Targeted tracing in a Software Product Line solution for managing telecommunication operator offers

Telecommunications market is continuously and rapidly evolving, in order to follow increasing demand and technological change. However, and despite the diversity of offers, their structure and the process of creation vary globally very little. That's why we thought about setting up a Product Line for the creation, management and maintenance of a telecommunication operators' offers.

Also, and due to the rapid evolution of the market and offers, traceability is a key element for the good maintenance of offers. It helps the operator be more reactive to identify similar offers, the impact of a new offer on existing ones, the components to be reused and those to be updated to gain in reactivity and time needed to get out a new offer on the market. This gain in time of response is very important for any telecom operator, not only to be more responsive in a very competitive environment, but also to meet the delay requirements of market regulator.

Considering all these elements, we decided to apply our traceability algorithm to a Product Line solution for managing telecommunication operator offers.

## 4.1 Offers Platform Description

Software Product Lines are recommended for large-scale systems, in order to bring a solution to the constraints of responsiveness to market developments; which is the case for telecom operators.

Indeed, the improvement of telecommunications that has become accessible and popular made the telecommunications market evolve exponentially, pushing operators to multiply and diversify their offerings. Also, the number of connected people increased, which made operators have a large customer base, and, therefore, must manage and propose offers based on the needs of its customers.

For all this, and knowing that the different offers of an operator generally meet the same constraints, including regulatory constraints, it is of great interest to set up a Software Product Line to manage the offers of a telecom operator. The figure Fig. 4 below describes this SPL solution.

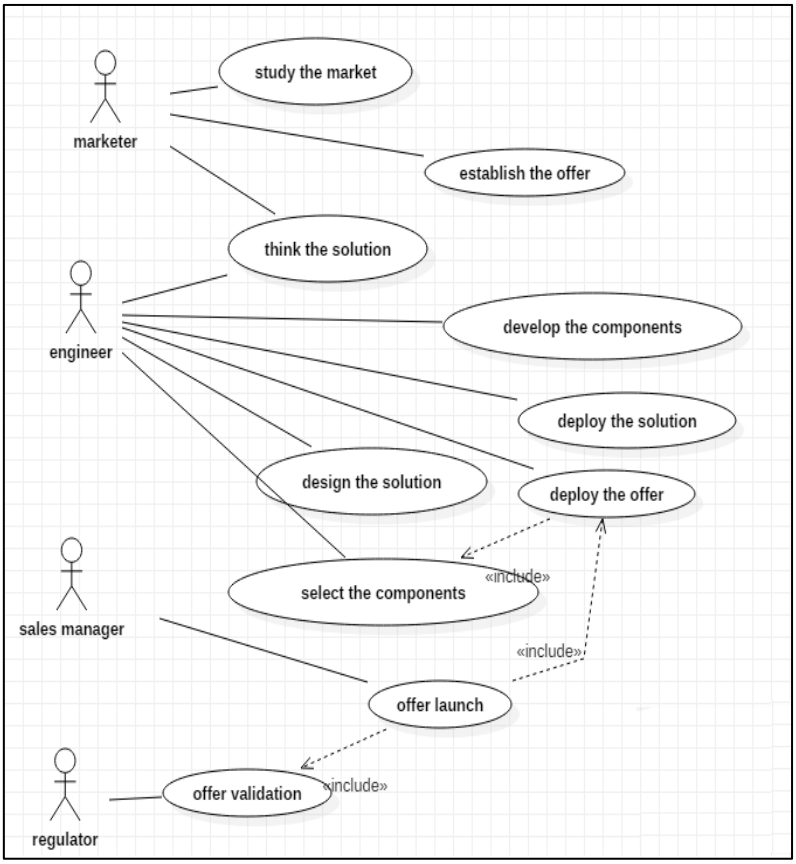


Fig. 4: Telecom operator offers management use cases

## 4.2 Our solution for tracing in Software Product Lines

To explain our tracing solution, we will focus on two main features: offer and validation.

The figure 5 below represents the feature-component diagram.

Let's consider the feature f1 "offer". This feature has properties that are set according to the offer to launch. Those properties include the segment or category of customers concerned by the offer (young people, seniors …), the data flow (voice, SMS or data), the duration of the offer (3 days, one week ...) and its price. Each of these properties has its own component. The dataflow component is a variation point, with voice, SMS and data as variables. Indeed, each of these types of flows has its own properties, such as the speed for the data, the number of hours for the voice and the type of SMS (SMS to a client or to an application) for the SMS.

Consider also the feature f2 "validation". This feature, intended to be used by the regulator to validate the commercial offer proposed by the operator, is based on the analysis of the price that will be proposed to customers for this offer, as well as the cost of the offer for operator and the cost of interconnection (fees charged by an operator A to an operator B for the use of his network, in particular during a call between a customer of the operator A and a customer of the operator B).

Figure 5 illustrates the relationship between these two features and the components connected to them.

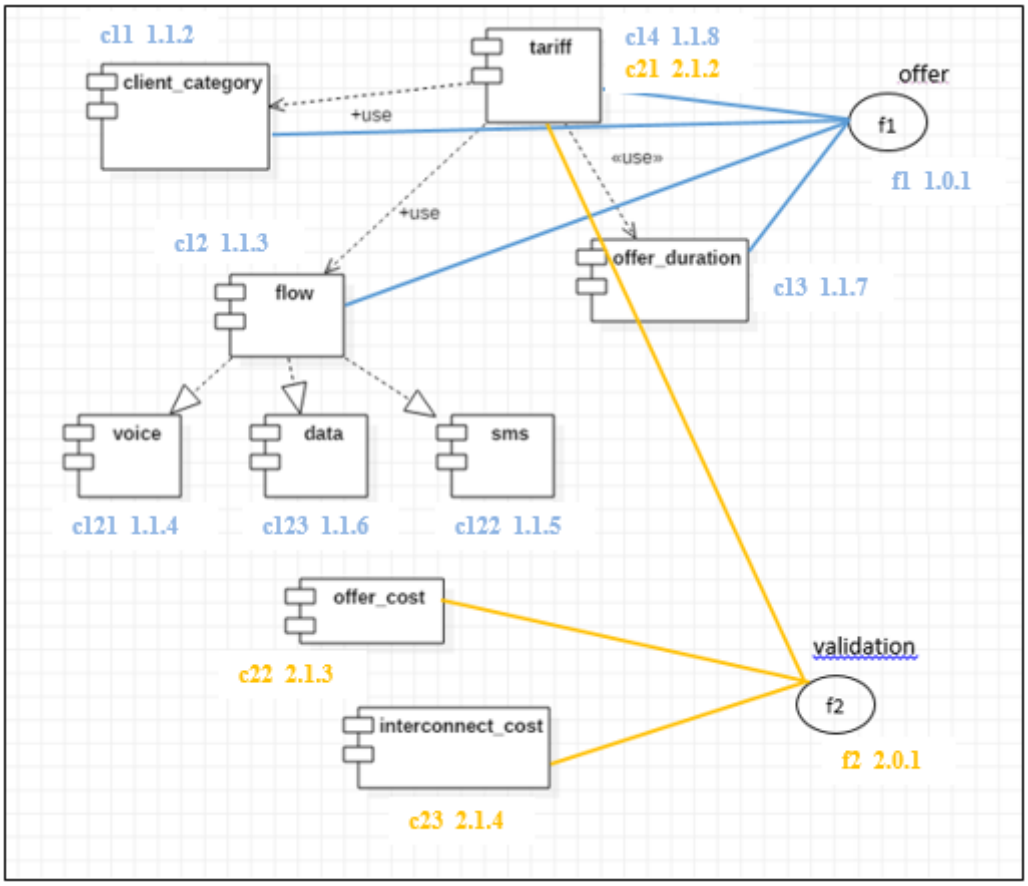


Fig. 5: Feature-component diagram

The first step of our algorithm is to define the domains. That is, the domains D1 and D2 respectively relating to the features f1 and f2. D1, with the identifier 1, has the following elements:

- feature f1 « offer »,
- component c11 : « client_category »,
- component c12, variation point : « flow »,
- component c121, variant of c12 : « voice »,
- component c122, variant of c12 : « SMS »,
- component c123, variante de c12 : « data »,
- component c13 : « offer_duration »,
- component c14 : « tariff »

Domaine D2, with the identifier 2, has the following elements:

- feature f2 : « validation »
- component c21 : « tariff »,
- component c22 : « offer_cost »,
- component c23 : « interconnect_cost ».

The second step of our algorithm is to assign addresses. In accordance with what was described earlier, we obtain the addresses as follows (Table 1):

Table 1: Assigned addresses

| Domaine | Element | Address |
|---|---|---|
| D1 | f1 | 1.0.1 |
| | c11 | 1.1.2 |
| | c12 | 1.1.3 |
| | c121 | 1.1.4 |
| | c122 | 1.1.5 |
| | c123 | 1.1.6 |
| | c13 | 1.1.7 |
| | c14 | 1.1.8 |
| D2 | f2 | 2.0.1 |
| | c21 | 2.1.2 |
| | c22 | 2.1.3 |
| | c23 | 2.1.4 |

Addresses 1.1.8 in D1 and 2.1.2 in D2 are related to the same component « tariff ». It is considered as a router that connects the two domains.

We consider a traceability of granularity = 1. We thus trace the links between the functions f1 and f2 and the components at the first level: c11, c12, c13 and c14 for the domain D1 and C21, c22 and c23 for the domain D2. An initial status equal to 1 is assigned to all these elements. If we come to add a second level of granularity, which is to trace also the variants c121, c122 and c123, we just need to broadcast the pair ($@_{c12}$, $status_{c12}$) to these variants, and to include them in any future broadcast. Here after the different scenarios.

**Feature removal:** Suppose that a validation before launching a new offer is no longer necessary. This involves deleting the f2 feature. In this case, the pair (2.0.1; 0) is distributed to the components c21, c22 and c23. c21 is also part of domain D1, it routes the information (2.0.1; 0) to feature f1 to inform it of the deletion of f2.

**Feature update:** The telecom operator can decide that an offer depends also on the customer's rate plan (example: a package can offer the customer 15h of communication and 2GB of data, or 30h of communications and 15GB of data, ...). In this case, a new component “rate_plan” is added to domain D1. Feature f1 now depends on a new item. This update of f1 is reported through the SPL by the diffusion of the pair (2.0.1; -1). As in the case of deletion, component c14 will route the information to feature f2.

**Change in a one address component:** suppose the offer of the operator no longer depends on the customer category. The component c11 should therefore be deleted. Therefore, and considering that we decided on a granularity = 1, the pair (1.1.2; 0) is diffused towards the feature f1 and components at the same level: c12, c13 and c14. Such removal modifies the logic of the offer, therefore the re f1 will be impacted. The previously illustrated algorithm for modifying a feature will have to be rolled.

**Change in a multi-addresses component:** Consider now the case where the base of calculation of the offers rate is modified. The "tariff" component is in this case updated and the pairs (1.1.8; -1) and (2.1.2; -1) are respectively communicated to the features f1 and f2. If this change of calculation in the tariff impacts the logic of the offer or the validation, then f1 and / or f2 will be impacted and the algorithm of modification of a feature will have to be rolled.

# 5. Conclusion

In this paper, we presented, as a part of our work on traceability in Software Product Lines, an algorithm for targeted tracing in SPLs. Indeed, through our researches, we found that in a complex large scale system such as SPLs, a targeted traceability has the most interesting Return On Investment in the mid and long term. Therefore, we worked on an algorithm that allows targeted and flexible tracing. We based this algorithm on dynamic routing protocols for IP networks.

We also presented a case study where we applied our tracing algorithm. The case is related to a Product Line solution for managing telecommunication operator offers. We described the proposed Product Line through its decomposition into features and components in order to apply the proposed algorithm.

However, some elements are still to be analyzed to go further in our research work, like the impact of environment heterogeneity and tracing granularity on the tracing algorithm.

# Références

Anquetil, N., Kulesza, U., Mitschke, R., Moreira, A., Royer, J. C., Rummler, A., et al., (2010). A model-driven traceability framework for software product lines. *Software & Systems Modeling*, 9, pp.427–451.

Bonaventure, O., (2010). *Computer Networking : Principles, Protocols and Practice*,

Cavalcanti, Y. C., do Carmo Machado, I., da Mota, P. A., Neto, S., Lobato, L. L., de Almeida, E. S., et al., (2011). Towards metamodel support for variability and traceability in software product lines. In *Proceedings of the 5th Workshop on Variability Modeling of Software-Intensive Systems - VaMoS '11*. New York, New York, USA: ACM Press, pp. 49–57.

Cisco (2010). Cisco ASA 5500 Series Configuration Guide using the CLI. Available at: https://www.cisco.com/c/en/us/td/docs/security/asa/asa82/configuration/guide/config/route_overview.pdf.

Cleland-Huang, J., Gotel, O. C., Huffman Hayes, J., Mäder, P., Zisman, A., (2014). Software traceability: trends and future directions. In *Proceedings of the on Future of Software Engineering*, pp. 55-69. ACM.

Cleland-Huang, J., Gotel, O., Zisman, A. eds., (2012). *Software and Systems Traceability*, London: Springer London.

Clements, P., Northrop, L., (2005). Software Product Lines. *Carnegie Engineering Institute*, pp.1–105.

Jirapanthong, W., Zisman, A., (2005). Supporting product line development through traceability. In *Proceedings - Asia-Pacific Software Engineering Conference, APSEC*. pp. 506–514.

Krueger, C., (2002). Eliminating the adoption barrier. *IEEE Software*, 19(4), pp.29–31.

Lindvall, M., Sandahl, K., (1996). Practical Implications of Traceability. *Software: Practice and Experience*, 26(10), pp.1161–1180.

Mcharfi, Z., El Asri, B., Dehmouch, I., Kriouile, A., (2015a). Measuring the impact of traceability on the cost of Software Product Lines using COPLIMO. *Proceedings of 2015 International Conference on Electrical and Information Technologies, ICEIT 2015*, pp.192–197.

Mcharfi, Z., El Asri, B., Dehmouch, I., Baya, A., Kriouile, A., (2015b). Return on Investment of Software Product Line Traceability in the Short, Mid and Long Term. In *Proceedings of the 17th International Conference on Enterprise Information Systems*. SCITEPRESS - Science and and Technology Publications, pp. 463–468.

Mcharfi, Z., El Asri, B., Kriouile, A., (2016). A Dynamic Tracing Model for Agile Software Product Lines Domain Engineering from Features to Structural Elements: An Approach Based on Dynamic Routing. *International Journal of Computer Science Issues*, 13(5), pp.94–101.

Medhi, D., Ramasamy, K., (2018). Networking and Network Routing: An Introduction. *Network Routing*, pp.2–29.

Northrop, L.M., (2002). SEI's software product line tenets. *IEEE Software*, 19(4), pp.32–40.

Ramesh, B., Jarke, M., (2001). Towards Reference Models for Requirements Traceability. *Software Engineering, IEEE Transactions on*, 27(1), pp.58–93.

Spanoudakis, G., Zisman, A., (2005). Software Traceability: A Roadmap. In *Handbook Of Software Engineering And Knowledge Engineering*. WORLD SCIENTIFIC, pp. 395–428.