Composition and Self-Adaptation of Service-Based Systems with Feature Models

Javier Cubo, Nadia Gamez, Lidia Fuentes, and Ernesto Pimentel

Dpto de Lenguajes y Ciencias de la Computación, Universidad de Málaga
{cubo,nadia,lff,ernesto}@lcc.uma.es

Abstract. The adoption of mechanisms for reusing software in pervasive systems has not yet become standard practice. This is because the use of pre-existing software requires the selection, composition and adaptation of prefabricated software parts, as well as the management of some complex problems such as guaranteeing high levels of efficiency and safety in critical domains. In addition to the wide variety of services, pervasive systems are composed of many networked heterogeneous devices with embedded software. In this work, we promote the safe reuse of services in service-based systems using two complementary technologies, Service-Oriented Architecture and Software Product Lines. In order to do this, we extend both the service discovery and composition processes defined in the DAMASCo framework, which currently does not deal with the service variability that constitutes pervasive systems. We use feature models to represent the variability and to self-adapt the services during the composition in a safe way taking context changes into consideration. We illustrate our proposal with a case study related to the driving domain of an Intelligent Transportation System, handling the context information of the environment.

Keywords: Service Composition, Self-Adaptation, Feature Models.

1 Introduction

Current pervasive systems are composed by a wide variety of services and devices. To reduce effort and costs, these systems may be developed using existing *Commercial-Off-The-Shelf* (COTS) components or (Web) services implemented by different vendors. Technologies such as Service-Oriented Architecture (SOA) [1] enable the building of fully working systems, as efficient as possible to improve the software reusability. The adoption of mechanisms for reusing software in pervasive systems has not yet become standard practice. This is because the use of pre-existing software requires the selection, composition and adaptation of prefabricated software parts. The discovery process aims to discover the most suitable services for a client request. The adaptation process solves, as automatically as possible, mismatch cases which may be given at the different interoperability levels among interfaces, while services are composed. Moreover, reusing software in critical domains (medical, automotive, aeronautics or security) is a difficult task, due to real complex problems such as guaranteeing high levels of efficiency and safety. For instance, in particular, the driving domain within the Intelligent Transportation Systems[1] (ITS) is a complex and safety critical environment. ITS are comprised of autonomous vehicles that can operate with minimum input from the driver. One of the critical aspects in this domain is the driver's interaction with the traffic environment. Therefore, these systems need to be developed taking into account the variability of the complex driving domain, which involves a dynamic adaptation to changing situations in the traffic environment, in order to fit the driver's safety and needs.

In addition to the wide variety of services (with different behaviours, components, elements, etc.), pervasive systems are composed of many networked heterogeneous devices (sensor nodes, smartphones, tablets, vehicles' on-board computers, or devices with RFIDs or cameras) with embedded software. Therefore, the heterogeneity can be present at any level. This can be addressed by using Software Product Line (SPL) engineering [2], which specifically focuses on variability management. SPLs aim to provide techniques for creating infrastructures that allow the rapid and systematic production of similar software systems, promoting the reuse of common core assets.

SOA and SPL approaches to software development share a common goal. They both encourage an organization to reuse existing assets and capabilities, rather than repeatedly redevelop them for new systems [3]. Then, we use these two complementary technologies to promote the safe composition in service-based systems. These systems have to be capable of handling changing situations during the composition, called *context changes*. Context information plays an important role in pervasive systems, to control their reaction depending on certain situations or to fit the user's needs. So, it is essential to manage contexts while composing services.

For this reason, as an initial attempt to solve these issues, we have developed and validated with several examples, the DAMASCo framework [4] based on SOA, which focuses on reusing services in pervasive systems accessed via their public interfaces, by means of context-aware service discovery, composition and adaptation.

However, DAMASCo still has some limitations as regards the service composition, since it does not take into account the variability of the services during the composition, which may also be changing depending on the contexts. We therefore need to address this new challenge of managing the variability of both services and contexts during the service composition process at runtime.

Feature Models (FM) [5] have been widely adopted by the SPL community to specify which elements, or *features*, of a family of products are common and which are variable. Then, a feature model permits the specification of where the variability is, independently of the core asset, and enables reasoning about all the different possible configurations of a family (corresponding to a service family in our case).

Therefore, in order to overcome the current restrictions of DAMASCo, in this work, as the main contribution, we propose to extend the DAMASCo framework with feature models to handle the runtime composition by means of service family discovery and self-adaptation when required. Thus, we use feature models to represent the variability of the services and to enable the service composition to dynamically reconfigure them when needed taking the context changes into account. To this end, we make use of Dynamic Software Product Lines (DSPLs) [6, 7], an emerging field that produces families of software products capable of adapting to

[1] `http://www.ewh.ieee.org/tc/its/` Accessed on 4 February 2013.

requirements that change at runtime. Following this paradigm, the service composition will be performed by selecting, at runtime, a specific configuration of the service family adapted to the context requirements. To illustrate our proposal, we use a case study related to the driving domain of an ITS, in which we compose pre-existing services and adapt them to satisfy a client request.

The remainder of the paper is organized as follows. In Section 2, we show the motivation behind our proposal, comparing it with related work. Section 3 presents the DAMASCo framework as the background to our approach and explains how it is extended with feature models. In Section 4, we define a mapping between the intermediate interface model used by DAMASCo and the feature models, and we apply our approach to a case study in the ITS domain. Section 5 presents a discussion of how our proposal overcomes some limitations of DAMASCo related to the variability of service-based systems. Finally, in Section 6 we outline some conclusions and plans for future work.

2 Related Work

In the last times there have been several approaches [8,11,12,13,14,15,16,17,18] that take advantage of using Dynamic Software Product Lines applied to Service Oriented Architectures. The rationale behind this is twofold: (i) the loosing coupling in SOAs can provide DSPLs with the technical underpinnings of flexible feature management; (ii) DSPLs can provide the modeling framework to undersign a self-adaptative SOA-based system by highlighting the relationships among its parts [8].

Following this convergence, in [8], with the purpose of reconfiguring service-oriented systems at runtime, the authors use the Common Variability Language[2] (CVL) to augment processes defined in the Business Process Execution Language (BPEL) [9] with variability, which makes it possible to easily generate a DSPL and they use a dynamic version of BPEL to manage and run it. Although this approach is very focused on two particular languages (CVL and DyBPEL), the feasibility of combining these two technologies is demonstrated. In our case, we exploit this combination for managing not only the dynamic reconfiguration, but also the safe composition and self-adaptation of services described using different business process languages without the need for any knowledge of variability languages. This is an important advantage because we define a mapping to automatically create the feature model representing a family of services from a business process specification. Our framework uses an intermediate interface model that can be generated from different platforms such as BPEL or Windows Workflow Foundation, WF [10].

With a similar motivation as our approach with regard to the necessity of SOA-based systems manage their inherent variability, in [11], SPL concepts to model SOA systems as service families are used. As we propose, the modeling of the SOA variability is performed by means of feature modeling and commonality/variability analysis technique. Particularly, SoaML[3] is extended with variability modeling notation. The main benefit of this approach is that different service variants can be

[2] `http://www.omgwiki.org/variability/doku.php` Accessed on 4 February 2013.
[3] `http://www.omg.org/spec/SoaML/` Accessed on 4 February 2013.

explicitly modeled, thus maximizing their reusability. However, this is not used to help the adaptation of the SOA systems at runtime. The authors study this task in [12], in which a member of the architecture can be dynamically adapted to a different member of the family at runtime. However, unlike our proposal, they do not deal with the service composition nor provide an automatic mapping to avoid SPL non-specialists have to handle with variability languages.

Montero et al. [13, 14] define a mapping between Feature Models and Business Process Model Notation[4] (BPMN) They provide a new semantic for feature models [13] in order to automate the family engineering process, obtaining the structure of a business process by means of model driven transformations. In [14], they propose the product evolution model for modeling runtime variability in business-driven systems to represent in which trigger events a business process evolves and how this evolution is managed. We also define a mapping, but with a different goal. Thus, we focus on representing the services, variability with feature models and using the DSPL paradigm to reconfigure them in order to make the service composition possible, by supporting the context changes.

Another approach that captures the variability in Business Process Modes is Provop [15], a framework for modeling and managing large collections of business process variants. In comparison to our proposal, they only mention allowing the dynamic reconfiguration of process variants at runtime as a future challenge, while we directly tackle this issue in this work.

In [16], the problem context-aware Dynamic Service-Oriented SPLs is tackled. The goal of this work is to simultaneously define at the same time a service-oriented and context-aware product derivation that monitors the context evolution in order to dynamically integrate the appropriate assets inside a running system where, as we propose, their target platforms follow the service-oriented approach. The authors address the self-adaptation problem, but they do not consider the service composition. In addition, as we have previously argued, our approach avoids forcing the definition of the services using feature models, since we propose an automatic mapping for this.

Finally, there are several approaches that use feature models to deal with the service composition [17, 18]. In [17] the matching between services during the composition is performed with feature models. The authors use feature modeling techniques to specify the variability of provided and required services, thus increasing the flexibility of the matching process. They define a mapping, but unlike us they do not provide any explanation of how the service self-adaptation process is performed. White et al. [18] use feature models to derive a new and correct service composition when a failure occurs. They demonstrate that leveraging feature models to automatically derive new service compositions, when a dependent service fails, the complexity of needing to model each individual error is eliminated. We also take advantage of this benefit, not only for composing the services when an error occurs but also for composing and self-adapting services that must work together correctly.

In summary, we take the demonstrated advantages of combining SOA and DSPL technologies by extending the DAMASCo framework with feature models, in order to manage the service variability during the discovery and composition of a service and to self-adapt the services when the context changing situations require it.

[4] `http://www.omg.org/spec/BPMN/` Accessed on 4 February 2013.

3 Our Approach

In this section, we first present the DAMASCo framework as the foundation of our approach which will then be described. It consists of extending this framework with feature models to support service composition whilst managing the variability safely.

3.1 Background: Service Reuse with DAMASCo

DAMASCo focuses on discovery, composition, adaptation and monitoring related to context-aware pervasive systems, where devices and applications dynamically find and use components and services from their environment.

It is based on SOA, the foal of which is to achieve loose coupling among interacting services, which is necessary and beneficial to the industry. However, SOA needs to be more agile and easier to model and reuse service applications. Modeling techniques, designing architectures, and implementing tools to support adaptation of the dynamic aspects in these systems represent new challenges in this research field. To address this, DAMASCo uses a model-based service-oriented architecture approach that makes the design, development and deployment of processes more agile. We focus on pervasive systems, such as ITS domain systems, composed of a service repository, users (clients requesting services), and a shared domain ontology.

DAMASCo adopts an expressive and user-friendly graphical notation based on transition systems, which reduces the complexity of modeling services. In addition, to discovering services, in DAMASCo, operation profiles of a signature refer to OWL-S concepts with their arguments and associated semantics. Once services have been discovered, in the case there are mismatch problems, an adaptor to solve problems is automatically generated using software adaptation. An adaptor is a third-party service in charge of coordinating services involved in the system. The whole process consists of a set of processes constituting the DAMASCo architecture, as shown in Fig. 1. The elements of DAMASCo have been implemented in Python as a set of tools which generate a framework integrated in the toolbox ITACA[5].

Service Interfaces. Each interface in DAMASCo is made up of a context profile, a signature, and a protocol specified as a transition system. At the user level, client and service interfaces can be specified by using: (i) context information in XML files for context profiles; we assume context information is inferred by means of the client's requests, in such a way that as a change occurs the new value of the context attribute is automatically sent to the corresponding service; (ii) WSDL[6] descriptions are used for describing the signatures in service-oriented platforms; and (iii) business processes defined in industrial platforms, as BPEL processes or WF workflows, for protocols that define service behavior. We consider clients and services implemented as business processes which provide the WSDL and protocol descriptions.

[5] Accessible at `http://itaca.gisum.uma.es`

[6] `http://www.w3.org/TR/wsdl` Accessed on 4 February 2013.

Model Transformation. First, interface specifications, which have not been previously transformed by the framework, are abstracted (Fig. 1, tag A). Context-Aware Symbolic Transition Systems (CA-STSs) are extracted from the BPEL services or WF workflows, which implement the client(s) and services, through our model transformation process [19]. We have defined CA-STS as an extension of Labelled Transition Systems (LTS) [20]. These intermediate models are graphical user-friendly, and CA-STS permits capturing contexts and their changes at runtime.

Semantic-Based Service Discovery. Then, a service discovery process (Fig. 1, tag B) finds out services satisfying the client's request, i.e., with compatible capabilities to the client requirements based on similar contexts, semantic matching of signature, and behavioural compatibility. Our process identifies mismatch situations using ontologies and synchronous product [20] to determine if adaptation is required or not.

Composition and Adaptation. If adaptation is not required, then the services of the systems are already deployed without having to adapt them, only performing the composition of them. Otherwise, a full service composition and adaptation process is executed (Fig. 1, tag C). Thus, an adaptation contract to solve mismatch problems is automatically obtained, and a CA-STS adaptor specification is generated [19]. Next, the corresponding BPEL or WF adaptor service is obtained from the CA-STS adaptor specification using our model transformation process (Fig. 1, tag D). Finally, the whole system is deployed, allowing the client and services to interact via the adaptor.

However, DAMASCo does not take into account the possible variability of the services when the matching is performed, nor the variability in the context changes. Therefore, to make our approach more useful, we propose extending DAMASCo by managing the service variability with feature models to safely handle the variability in the composition and self-adaptation of the services at runtime.

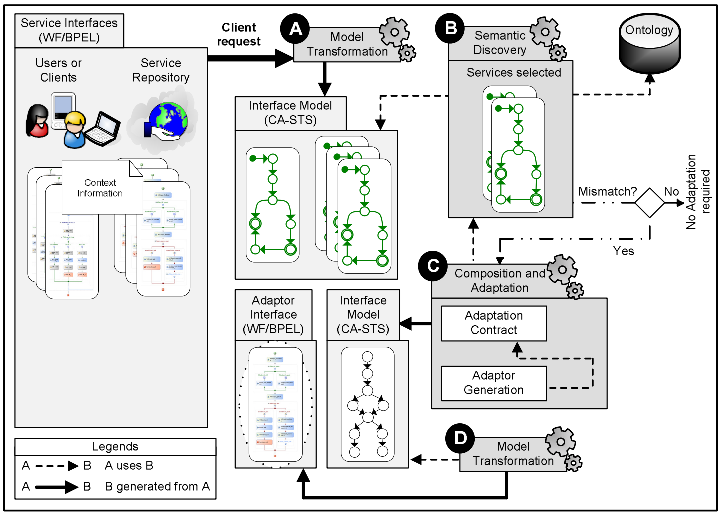


Fig. 1. DAMASCo framework architecture

3.2 Adding Feature Models to Support Safe Composition

Here, we explain how we integrate the feature models into DAMASCo, as shown in Fig. 2, in order to deal with the variability of the services during the composition. First though we briefly describe some necessary concepts of feature models.

Formally, a Feature Model [5] is a hierarchical decomposition of features to specify which elements of a family of products are common, which are variable and the reasons why they are variable, i.e., whether they are alternative or optional elements. Furthermore, apart from the relationship between the features in the diagram (called *tree constraints*), a feature dependency analysis can identify dependencies between features (called *cross-tree constraints*). Examples of such dependencies are the mutual dependency and mutual exclusion relationships. A feature model configuration is the selection of a set of features belonging to the feature model. A configuration is valid if all the features are contained in the configuration, and the non-selection of all other specific features is allowed by the feature model [21]. Thus, a valid configuration must satisfy the tree and cross-tree constraints. In our case, every valid configuration represents a potential service, but only a subset of all these possibilities are already deployed in the service repository.

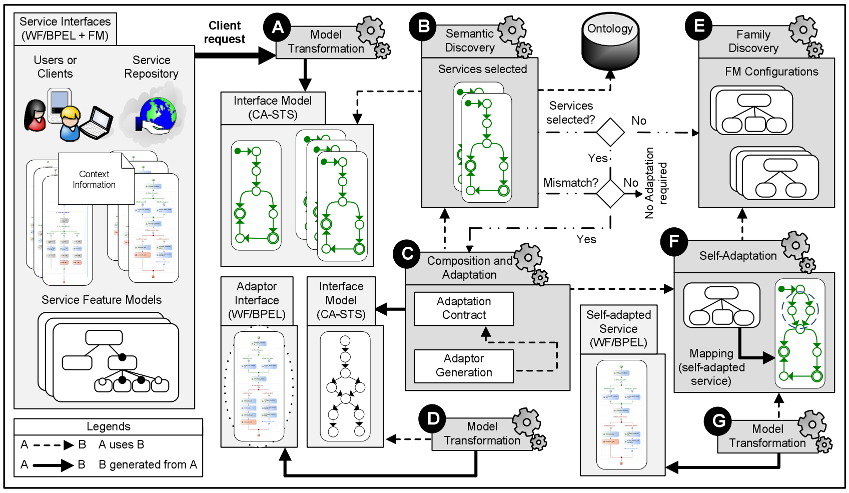


Fig. 2. FM-DAMASCo framework architecture

Firstly, in addition to the BPEL/WF service descriptions, we have a feature model for each service that describes its variability, i.e., the service family. Then, each business process corresponds with a valid configuration of the feature model that represents it, i.e., a specific product of the service family. These feature models can be designed by a user developer. Nevertheless, if we do not want to force them to have all the services represented by feature models, we can automatically generate the feature model that contains the variability corresponding to a specific service instance using the mapping that we will describe in Section 4. We focus on the representation of the service variability with respect to the context, for instance, a navigation service family may have traffic management as an optional feature, and so, this feature can be selected in a configuration where road traffic information at real time has to be considered as part of the context.

After a client executes a request, both the DAMASCo model transformation and the semantic-based service discovery process (Fig. 2, tags A and B) are activated. The semantic discovery, as previously described, tries to find the proper services for the client's request. However, due to the high variability of the services, it is possible that although a service instance of the repository matches the request, a small variation of a service could be enough to match exactly. For example, let us imagine there is no deployed navigation service with a traffic monitoring component, but there is a traffic feature in the navigation service family. In this case we need to incorporate a new process to the DAMASCo framework, that we call service family discovery (Fig. 2, tag E). We use the feature models to find a new matching, regarding the features that may suit a certain context. If the family discovery process finds the service family with the adequate feature for the context, a new valid configuration of that family containing this feature is automatically created by our feature modeling tool, Hydra[7]. We want to highlight the new configuration is valid, so, both tree and cross-tree constraints specifying restrictions between the components and context are satisfied. Hydra cannot create a configuration that does not satisfy the restrictions, so the reconfiguration is done safely, as invalid configurations are not possible.

Once the configuration representing the particular services is automatically generated by Hydra, the new service self-adaptation process added to DAMASCo (Fig. 2, tag F) is executed. Then, the CA-STS (intermediate interface model) corresponding with this feature model configuration is automatically created using the mapping defined in Section 4. Finally, this new CA-STS interface is transformed into a WF/BPEL process, following the procedure explained in Section 3.1, which is composed with the other services to satisfy the request.

4 Self-Adaptation Using Feature Models

To illustrate our proposal, we firstly present a driving domain scenario of an ITS. We assume some services have been implemented, and we manage the service composition handling the changing situations of the environment. Secondly, we define a mapping to generate the correspondences between the CA-STS model representing a service interface and the feature model of a service family. Lastly, we apply the mapping and the self-adaptation process to our case study.

4.1 Case Study: A Driving Domain

Our example consists of a driver and a service repository, whose services may be composed to get a specific purpose. The driver can perform a navigation request, and depending on context and service variations, the system must be adapted to work correctly in any situation. Services such as message console, on-board entertainment, navigation, maps, traffic management, weather information, or POI notifications, are

[7] Accessible at `http://caosd.lcc.uma.es/spl/hydra`

some of the applications in vehicles [22]. In our driving scenario, we have implemented (in BPEL and WF) the following services: message console, navigation, maps and traffic services. Figure 3 shows the transition systems (represented with our CA-STS interface model) corresponding to the implemented services, obtained through the model transformation process (see Figure 1), in which we have abstracted parameters to simplify the interfaces.

The driver uses the message console to request the navigation to a specific destination, which in turn interacts with the navigation service that is in charge, using the maps service, of calculating the route to the destination. The context information detailed in the context profile of each service means the service requires such contexts, and they have to be considered during the composition of the system at runtime. For example, the message console service automatically obtains both the location and the language of the driver (*loc* and *lang* contexts) from the GPS and on-board computer settings, respectively. On the other side, the driver will indicate the context info related to the route type (*route* context), as well as whether he/she wants to avoid the toll road or not (*toll* context) and with traffic monitoring or not (*traffic* context). These driver preferences may change at runtime. Default values for these contexts are used if the driver does not specify them. DAMASCo performs the composition process of this scenario, in which CA-STS service interfaces are synchronised through an adaptation process that solve any mismatch problems during the composition. As explained in Section 3, we use a semantic-based discovery that uses an ontology to search services that match with the client's request. We have generated a driving domain ontology for our example using Protégé 4.0.2.

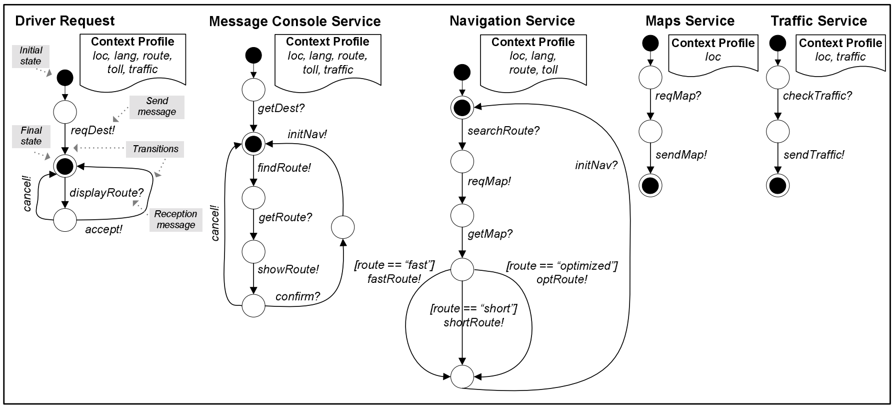


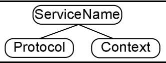
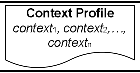
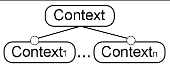
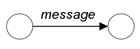
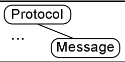
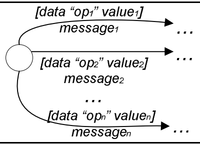
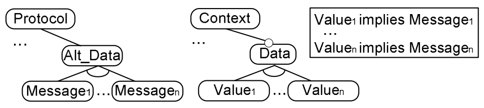
Fig. 3. Interface models of the driver request and the services for the driving domain scenario

Once DAMASCo performs the composition, it can control dynamic context changes, by capturing and handling such changes, and it simulates the dynamic update of the environment according to the context changes at runtime. Now, let us imagine the driver decides that the route be calculated avoiding any possible traffic problems at runtime. Therefore, in this case the traffic context needs to be considered in the services participating in the composition. In addition, the semantic discovery process cannot find a navigation service which checks the traffic management. Therefore, the FM-DAMASCo uses the discovery family process, which finds a feature traffic, among the navigation service family, and the self-adaptation of the navigation service for adding this feature to it. This will be explained in Section 4.3, but first, we present the mapping between CA-STS models and feature models.

4.2 Mapping between Interface Models and Feature Models

In order to avoid new feature models having to be defined for services already deployed in a specific pervasive system, we define a mapping between the CA-STS interface model and the feature model (Table 1). This mapping allows the automatic generation of the feature model that corresponds with a service defined in BPEL or WF. Then, we also use the model transformation between WF/BPEL and CA-STS. As shown in Table 1, for each service we create a new feature model with the service name as the root feature. This root feature has two mandatory children: protocol and context. The context feature has all the context items defined in the context profile as optional children. They will be the contexts that may be considered during the service execution. The protocol will contain as children, the features with the ordered message names according to the message sequence defined in the corresponding CA-STS. Then, for every message, which implies a transition in the CA-STS model, a new mandatory feature message must be added as a child of the protocol feature.

Table 1. Mapping between CA-STS Model and the Feature Model of the service family

CA-STS Interface Model	**Feature Model**
ServiceName	
**Context Profile** *context₁, context₂,…, contextₙ*	
	
 [data "op₁" value₁] message₁ ... [data "op₂" value₂] message₂ ... ... [data "opₙ" valueₙ] messageₙ ...	 Protocol ... Alt_Data Message₁ ... Messageₙ Context ... Data Value₁ ... ValueₙValue₁ implies Message₁ ... Valueₙ implies Messageₙ

Finally, we map the alternative sequences (such as *ifelse* or *pick* activities in BPEL or WF) represented in the CA-STS interface as different branches or transitions. Since these alternatives will send or receive several messages depending on different values of data, we add a mandatory child of the protocol feature that will contain alternative XOR features for every message. Furthermore, we must add a cross-tree constraint with a mutual dependency of the value that implies a message (e.g., *$Value_1$ implies $Message_1$*). In addition, if this data coincides with a context item, then we add the values as alternative XOR features of the data context.

Apart from the purpose of using the mapping to represent the services without a previous correspondence with a feature model, this mapping is also used to help the self-adaptation of services when required during the composition. Following the DSPL paradigm, the runtime self-adaptation can be defined in terms of replacing the current feature model configuration for a new configuration adapted to the current requirements. As described in Section 3.2, in the case that the semantic discovery process does not find the service instance that matches with the client request, the service family discovery will try to find, from among the family, a variation of the existent services that better fits the request. Then, in our approach, applying the DSPL paradigm for self-adaptation during composition also means having to replace a feature model configuration with another one. In this case, we replace the configuration representing the service which better matches with the request with another configuration with a small variation to exactly match the request.

Therefore, our process uses both configurations (the previous and the new one [23, 24]), the feature model, the CA-STS interface of the service that fits better, and the mapping to automatically create the new CA-STS which represents the adapted service. In the next section, we detail this process and the mapping over our example.

4.3 Applying our Approach in ITS

Figure 4 shows the feature model of the navigation service. As defined in the mapping it is composed of the protocol and context features. The protocol contains the mandatory features that have to be in all the navigation services (the ones that appear in the navigation service of Figure 3) and several optional features, such as the traffic management or the point of interest (POI) alerts. Furthermore, some features have several XOR alternative children, like the type of route to be calculated (fast, short or optimized route). The context contains all the possible (optional features) context items that the navigation system may consider, such as location, traffic, weather, and so on. For the sake of simplicity, we have only represented (in Figure 4) the possible context values for the type of route context to calculate the route according to the driver preferences, and for the traffic context to indicate whether the driver requires traffic monitoring (traffic true) or not (traffic false).

As explained in the mapping definition, there are also dependencies between the context values and the alternative features in the protocol. For instance, *Fast implies FastRoute* means that if the context feature *Fast* is selected in a valid feature model configuration, then the protocol message feature *FastRoute* must also be selected. In this way, we avoid creating invalid configurations where the context variations are not satisfied by the protocol. This is verified every time we create a new configuration during the self-adaptation process in order to carry out a safe reconfiguration.

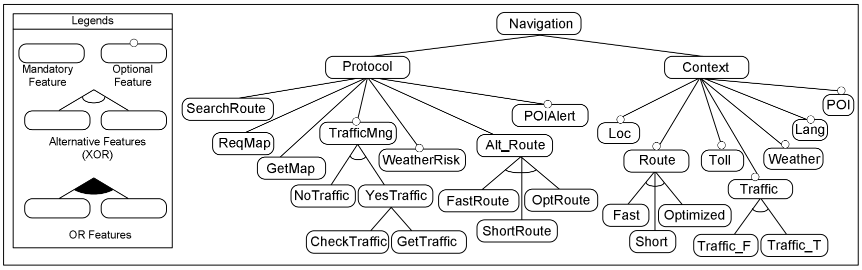


Fig. 4. Feature Model for the navigation service

The self-adaptation of the navigation service to include the traffic management is depicted in Figure 5. Firstly, the navigation feature model configuration represents the corresponding navigation service shown in Figure 3. This configuration does not consider the traffic monitoring in its context, so the protocol calculates the route without taking traffic incidences into account. Furthermore, other contexts (such as the notification of nearing points of interest and the calculation of the route taking into account weather warnings) are not considered in this configuration, so their correspondent optional features are removed. In the case that the driver wants to incorporate the traffic context into the navigation service, this service can be adapted as that context and feature are contemplated in the navigation family (Figure 4). At feature model level, this adaptation consists of adding the optional features (and its children) related to the traffic not selected in the previous configuration, in a new configuration, as observed in the navigation feature model adapted configuration (shown in green in Figure 5). Then, our self-adaptation process uses the CA-STS interface of the previous navigation service, the navigation FM configuration, the navigation FM adapted configuration, and the mapping to automatically generate the CA-STS interface model of the navigation service adapted, which already consider the traffic management (corresponding to the part of the CA-STS interface in green, surrounded by a dashed blue circle in Figure 5). The adapted navigation service will be composed with the rest of the services of our driving scenario, including the traffic service in the composition, in the same way as explained in Section 4.1. With this application, we illustrate how by using our approach we obtain a safe reuse based on a self-adaptation of the navigation service, with the purpose of fulfilling a request that a priori would have not been satisfied by any other existing service of the repository.

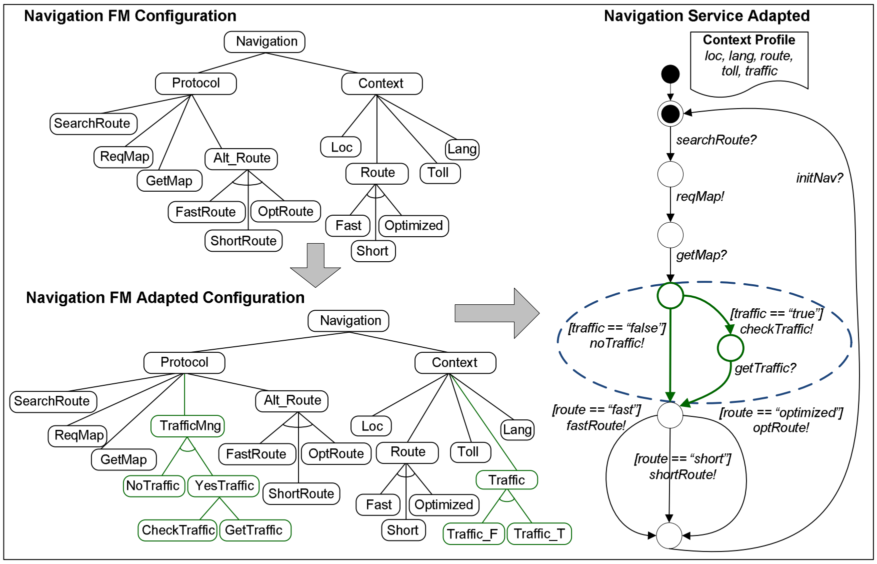


Fig. 5. Self-Adaptation of the navigation service

5 Discussion

In this section, we discuss (i) the benefits and drawbacks, (ii) the main contributions for the ITS domain, and (iii) other possible applications of our approach.

Benefits and Drawbacks. The main benefit of our proposal is that, using the SPL approach, we can significantly increase the number of client requests satisfied in a repository with a relatively small number of services deployed. For instance, in our navigation service family we have three optional features (Figure 3) in the protocol. This means that these features can be present or not in a service valid configuration. Selecting or unselecting these three optional features we have a total of 2^3=8 valid configurations, i.e., 8 different services for this family. So, although there will be deployed, e.g., 2-3 different navigation service instances in the repository, with our approach we can carry out 8 different kinds of requests for this service. These numbers are only considering the optional features of a family. But if we consider the variable (OR and XOR) features, the number of configurations increases greatly. Table 2 shows the number of possible valid configurations with respect to the number of family services for a specific domain and considering the average of the variable parts per service. Thus, 10 services working together could satisfy (2^{10}-1)=1023 different potential client requests. Nevertheless, if as we propose in this work, instead of 10 single services we have 10 service families with an average of 2^3=8 possible configurations per service, then we can satisfy 8*1023=8184 requests. Then, using our approach, in this small service repository with a few members per family, we will increase the number of possible requests satisfied by more than 7000. Furthermore, a real repository for a specific domain (e.g., the driving domain) can have 20 services with an average of 6 variations (optional and alternative features) per service. In this case, we have a total number of 1280 possible valid configurations and the number of possible client requests satisfied by these configurations increases exponentially.

Table 2. Number of possible valid services configuration

Service Families	Variations per Service	Valid Configurations
10	3	80
10	6	640
20	6	1280
30	10	30270

Obviously, this entails an aggregate cost, as to set up a SPL infrastructure may require great effort and the designing of 20 service families is a non-trivial task. Although the defined mapping can be used to automatically create a feature model from an interface model, this will represent a single service and not a whole family. Then, the variability must be added to the feature model by hand. However, though the definition of the service families may have an initial cost, as soon as individual services start to be automatically adapted in order to satisfy different client requests, which would otherwise require the implementation of new services, this initial effort becomes cost-effective. Therefore, the adoption of the SPL approach is justified in repositories where the number of services and possible variants per service is not very small. For instance, for 10 services with 3 variations per service, we can satisfy more than 7000 requests more than in the case of 10 single services. However, if we have a repository with 3 services and 2 variations per service, we only satisfy 5 requests more using our approach. In this case, the effort to build the family is not rewarded. Another example which involves a greater cost that provided benefits is in static environments where not many client requests are made during the system execution. In this case, it is pointless to have big service families with configurations that will never be used. We must therefore look for a compromise between the repository size and the necessity of defining the service families considering the benefits obtained.

Finally, we wish to stress that the service family discovery process is performed at family level not at configuration level. Therefore, in order to search for matches, the process does not have to look for the thousands of .xml files representing all the configurations (30270 in the case of 30 services), but only in the 30 .xml files representing the families, which is a fast task for current computers.

Contributions in the ITS Domain. Traditionally, when a vehicle is designed and manufactured, it is given a specific set of hardware and software components. This is a disadvantage in case new applications have to be incorporated to the vehicle, reducing the cost-effectiveness of the implementation and maintenance of vehicular software. For this reason, AUTOSAR[8] (AUTomotive Open System ARchitecture) is a worldwide development cooperation of car manufacturers, suppliers and other companies such as software industry. The main purpose is to provide a basic infrastructure to help develop vehicular software. Nevertheless, AUTOSAR currently only delivers the standard specifications not an implementation of the basic software.

Aspects such as electronic tolling, road safety, the user interface, and the provision of information to the driver, are crucial in the vehicular environment. In order to achieve these objectives, it is essential to develop a correct architecture for the definition of services. Some work [25] has already been worked on the creation of a service-oriented architecture for an on-board computer, by using the composition as OSGi technology and the development of system management information through web and distributed environments. In addition, techniques to address the variability of the complex driving domain have to be considered, allowing the adaptation to changing situations in traffic environments and to meet the drivers' safety and needs.

The generation of an architecture and the implementation of the services is beyond of scope of this work. Our proposal is complementary to these two different efforts (work in [25] and the AUTOSAR initiative), since we tackle the reuse and maintenance of previously implemented services with the main goal of facilitating the handling the variability, in this case, of the driving domain. And we do this by means of self-adaptation mechanisms based on SOA and SPL paradigms.

Other Applications. In this work, we have focused on applying the DSPL approach to self-adapt the services when is required during the context-aware composition. But, as we have mentioned, other applications can be also tackled with our approach, such

[8] `http://www.autosar.org/` Accessed on 4 February 2013.

as service dynamic reconfiguration or evolution. On the one hand, ITS domain is an example of systems that should be able to adapt their devices to some context changes with minimum human intervention, and so a given kind of dynamic self-adaptation is necessary to adapt them to context changes, such as network degradation or sudden events. In this sense, DAMASCo uses the DSPL approach by replacing the current FM configuration for a new one adapted to the context change, as explained in [26]. On the other hand, ITS is also an outstanding example of a modern system that is in permanent evolution, as new devices, technologies or facilities continuously appear. This means it is desirable to have a mechanism that helps with the propagation of evolution changes in deployed systems. For this task, DAMASCo will use the results of our previous work for managing the evolution of product families [23, 24] and together with the mapping it will be able to evolve with new components of services already deployed. With these two applications, apart from the context-aware composition, we demonstrate the wide range of applicability of our approach based on SOA and SPL technologies working together.

6 Conclusions

In this paper, we have illustrated the need to handle the variability during the service composition in pervasive systems. Our proposal to address this challenge is based on both SOA and DSPL paradigms. Thus, we extend our DAMASCo framework with feature models to represent the variability and to dynamically reconfigure services in a safe way according to context change situations. Specifically, we have developed two new processes in DAMASCo: a service family discovery and a self-adaptation mechanism, which have been described throughout the paper. We have implemented our approach in a scenario of the ITS domain, and we have discussed the benefits, drawbacks, contributions, and other possible applications it could have.

As regards future work, we are currently working on the other two applications we discussed in the previous section. We plan to define a model-driven process to switch from one running service configuration to another by executing a plan in DAMASCo in order to self-adapt the services to context changes at runtime. In addition, to perform evolution we need to define how modifying or aggregating new behaviour (not previously contemplated for the family) into already existing services.

Acknowledgements. Work partially supported by the projects TIN2008-05932, TIN2008-01942, TIN2012-35669, TIN2012-34840 and CSD2007-0004 funded by Spanish Ministry of Economy and Competitiveness and FEDER; P09-TIC-05231 and P11-TIC-7659 funded by Andalusian Government; and FP7-317731 funded by EU.

References

1. Erl, T.: Service-Oriented Architecture (SOA): Concepts, Technology, and Design. Prentice-Hall, Englewood Cliffs (2005)
2. Pohl, K., Böckle, G., Linden, F.: Software Product Line Engineering – Foundations, Principles, and Technique. Springer, Heidelberg (2005)

3. Krut, R., Cohen, S.: Service-Oriented Architectures and Software Product Lines - Putting Both Together. In: Proc. of SPLC 2008, p. 383. IEEE Computer Soc., Los Alamitos (2008)
4. Cubo, J., Pimentel, E.: DAMASCo: A Framework for the Automatic Composition of Component-Based and Service-Oriented Architectures. In: Crnkovic, I., Gruhn, V., Book, M. (eds.) ECSA 2011. LNCS, vol. 6903, pp. 388–404. Springer, Heidelberg (2011)
5. Lee, K., Kang, K.C., Lee, J.: Concepts and Guidelines of Feature Modeling for Product Line Software Engineering. In: Gacek, C. (ed.) ICSR 2002. LNCS, vol. 2319, pp. 62–77. Springer, Heidelberg (2002)
6. Hallsteinsen, et al.: Dynamic Software Product Lines. Computer 41(4), 93–95 (2008)
7. Shen, L., Peng, X., Liu, J., Zhao, W.: Towards Feature-Oriented Variability Reconfiguration in Dynamic Software Product Lines. In: Schmid, K. (ed.) ICSR 2011. LNCS, vol. 6727, pp. 52–68. Springer, Heidelberg (2011)
8. Baresi, L., Guinea, S., Pasquale, L.: Service-Oriented Dynamic Software Product Lines. Computer 45(10), 42–48 (2012)
9. Andrews, T., et al.: Business Process Execution Language for Web Services (WSBPEL). Systems, IBM, Microsoft, SAP AG, and Siebel Systems (2005)
10. Scribner, K.: Microsoft Windows Workflow Foundation: Step by Step. Microsoft (2007)
11. Abu-Matar, M., Gomaa, H.: Variability Modeling for Service Oriented Product Line Architectures. In: Proc. of SPLC 2011, pp. 110–119. IEEE Computer Soc., Los Alamitos (2008)
12. Gomaa, H., Hashimoto, K.: Dynamic Software Adaptation for Service-Oriented Product Lines. In: Proc. of SPLC Workshops 2011. ACM (2011)
13. Montero, I., Pena, J., Ruiz-Cortes, A.: From Feature Models to Business Processes. In: Proc. of SCC 2008, pp. 605–608. IEEE Computer Soc., Los Alamitos (2008)
14. Montero, I., Peña, J., Ruiz-Cortes, A.: Representing Runtime Variability in Business-Driven Development Systems. In: Proc. of ICCBSS 2008, February 25-29, p. 241. IEEE Computer Soc., Los Alamitos (2008)
15. Hallerbach, A., Bauer, T., Reichert, M.: Capturing Variability in Business Process Models: The Provop Approach. Journal of Software Maintenance and Evolution: Research and Practice 22(6-7), 519–546 (2010)
16. Parra, C., Blanc, X., Duchien, L.: Context Awareness for Dynamic Service-Oriented Product Lines. In: Proc. of SPLC 2009, pp. 131–140 (2009)
17. Naeem, M., Heckel, R.: Towards Matching of Service Feature Models based on Linear Logic. In: Proc. of the 1st Workshop on Services, Clouds, and Alternative Design Strategies for Variant-Rich Software Systems (SCArVeS) Co-Located with SPLC 2011 (2011)
18. White, J., Strowd, H.D., Schmidt, D.C.: Creating Self-Healing Service Compositions with Feature Models and Microrebooting. Int. Journal of Business Process Integration and Management 4(1), 35–46 (2008)
19. Cubo, J., Canal, C., Pimentel, E.: Context-Aware Composition and Adaptation Based on Model Transformation. Journal of Universal Computer Science 17(15), 777–806 (2011)
20. Arnold, A.: Finite Transition Systems. International Series in Computer Science. Prentice-Hall, Englewood Cliffs (1994)
21. Batory, D.: Feature Models, Grammars, and Propositional Formulas. In: Obbink, H., Pohl, K. (eds.) SPLC 2005. LNCS, vol. 3714, pp. 7–20. Springer, Heidelberg (2005)
22. Lee, J., Kotonya, G., Robinson, D.: A Negotiation Framework for Service-Oriented Product Line Development. In: Edwards, S.H., Kulczycki, G. (eds.) ICSR 2009. LNCS, vol. 5791, pp. 269–277. Springer, Heidelberg (2009)

23. Gamez, N., Fuentes, L.: Software Product Line Evolution with Cardinality-Based Feature Models. In: Schmid, K. (ed.) ICSR 2011. LNCS, vol. 6727, pp. 102–118. Springer, Heidelberg (2011)
24. Gamez, N., Fuente, L.: Architectural Evolution of FamiWare using Cardinality-Based Feature Models. Journal of Information and Software Technology 55(3), 563–580 (2013)
25. Santa, J., Úbeda, B., Gómez-Skarmeta, A.F.: A Multiplatform OSGi Based Architecture for Developing Road Vehicle Services. In: Proc. of CCNC 2007, pp. 706–710. IEEE Computer Soc., Los Alamitos (2007)
26. Gamez, N., Fuentes, L., Aragüez, M.A.: Autonomic Computing Driven by Feature Models and Architecture in FamiWare. In: Crnkovic, I., Gruhn, V., Book, M. (eds.) ECSA 2011. LNCS, vol. 6903, pp. 164–179. Springer, Heidelberg (2011)