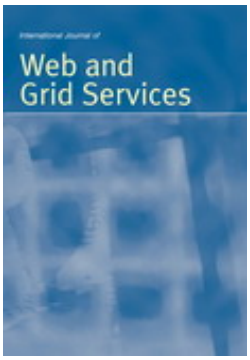
INDERSCIENCE PUBLISHERS
Linking academia, business and industry through research



International Journal of Web and Grid Services

ISSN online: 1741-1114 - ISSN print: 1741-1106
https://www.inderscience.com/ijwgs

A feature-driven variability-enabled approach to adaptive service compositions

Chang-ai Sun, Zhen Wang, Zaixing Zhang, Luo Xu, Jun Han, Yanbo Han

DOI: 10.1504/IJWGS.2023.10054495

Article History:

Received:	06 April 2022
Last revised:	22 October 2022
Accepted:	14 November 2022
Published online:	06 March 2023

A feature-driven variability-enabled approach to adaptive service compositions

Chang-ai Sun*, Zhen Wang and Zaixing Zhang

School of Computer and Communication Engineering,
University of Science and Technology Beijing,
Beijing, 100083, China
Email: casun@ustb.edu.cn
Email: zhenwang@xs.ustb.edu.cn
Email: 13051508603@163.com
*Corresponding author

Luo Xu

North China Institute of Computing Technology,
Beijing, 100083, China
Email: xdotstone@hotmail.com

Jun Han

Department of Computer Science and Software Engineering,
Swinburne University of Technology,
Hawthorn, VIC 3122, Australia
Email: jhan@swin.edu.au

Yanbo Han

Beijing Key Laboratory on Integration and
Analysis of Large-Scale Stream Data,
North China University of Technology,
Beijing, 100144, China
Email: yhan@ict.ac.cn

Abstract: Service compositions are widely used to construct complex applications. Due to the frequent changes of environment and requirements, service compositions need to be adaptable enough. In this work, we propose a feature-driven variability-enabled adaptive service composition approach to systematically treat the variability in the full life-cycle of service compositions. Specifically, the feature model is introduced to represent common and variable requirements and drive the variability design of service compositions. An abstract service composition model is used to define the variable business process. Rules and algorithms are then defined to transform the feature model to the abstract service composition model, from which different process instances are derived on demand to meet different requirements. We have developed a prototype tool to facilitate and automate

our approach as much as possible. Finally, a case study is conducted to demonstrate the proposed approach and validate its effectiveness and efficiency.

Keywords: variability management; adaptive service compositions; abstract service composition model; feature model; model transformation.

Reference to this paper should be made as follows: Sun, C-a., Wang, Z., Zhang, Z., Xu, L., Han, J. and Han, Y. (2023) 'A feature-driven variability-enabled approach to adaptive service compositions', *Int. J. Web and Grid Services*, Vol. 19, No. 1, pp.79–112.

Biographical notes: Chang-ai Sun is a Professor in the School of Computer and Communication Engineering, University of Science and Technology Beijing. Before that, he was an Assistant Professor at Beijing Jiaotong University, China, a Post-doctoral Fellow at the Swinburne University of Technology, Australia, and a Post-doctoral Fellow at the University of Groningen, The Netherlands. He received his Bachelor's in Computer Science from the University of Science and Technology Beijing, China, and PhD in Computer Science from the Beihang University, China. His research interests include service-oriented computing, software testing, and program analysis.

Zhen Wang is a PhD student in the School of Computer and Communication Engineering, University of Science and Technology Beijing, China. She received her Bachelor's in Computer Science and Technology from the University of Science and Technology Beijing, China. Her current research interests include service-oriented computing and internet of things.

Zaixing Zhang is a Master's student in the School of Computer and Communication Engineering, University of Science and Technology Beijing. His current research interest is service-oriented computing.

Luo Xu is a researcher in the North China Institute of Computing Technology. He received his PhD in Computer Software and Theory from the Beihang University, China. His research interests include software architecture and software testing.

Jun Han is a Professor in the Department of Computer Science and Software Engineering, Swinburne University of Technology, Australia. He received his PhD in Computer Science from The University of Queensland, Australia. His research interests include service computing and software architecture.

Yanbo Han is a Professor at the North China University of Technology. He is also the Director of Beijing Key Laboratory on Integration and Analysis of Large-Scale Stream Data. He received his PhD in Computer Science from the Technical University of Berlin, Germany. His research interests include service computing and data analysis.

1 Introduction

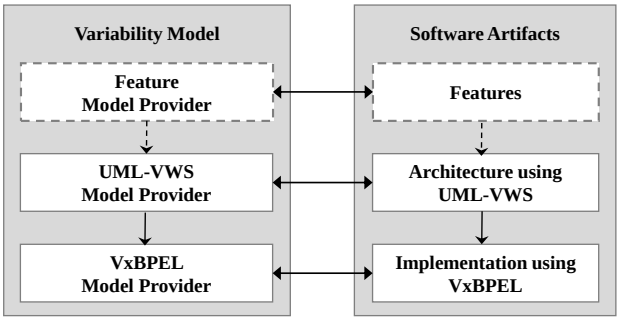
Service-oriented architecture (SOA) has become a mainstream application paradigm in distributed and heterogeneous environments (Lemos et al., 2015). Increasingly, companies, such as Netflix and Google, employ this architecture to improve the development efficiency and scalability of their business systems (Bai et al., 2009). In this context, the complex business process can be implemented by coordinating distributed services according to specifications, which are known as service compositions (Peltz, 2003). At the same time, the operating environment and requirements of the services may change dynamically and frequently. For instance, one or more component services may become unavailable at run-time or user requirements may suddenly change for some unexpected reason. Accordingly, service compositions need to be adaptable to these changes to provide the expected functionalities continuously (Xiao et al., 2011).

To address the above problem, several approaches have been investigated over the past decades. Among them, a class of representative approaches are based on rules, such as those via proxy (Ezenwoye and Sadjadi, 2007), aspect-oriented programming (AOP) (Krishnamurty et al., 2013), and context (Urbieta et al., 2017; Valderas et al., 2022), which normally predefine event rules at the design phase and trigger them at run-time to achieve the adaptability of service compositions. They mainly focus on changes of business process design and implementation without explicitly taking into account the full lifecycle of changes. Recently, researchers have also explored artificial intelligence (AI) techniques, e.g., AI planning (Bashari et al., 2018), reinforcement learning (Wang et al., 2020; Gharineiat et al., 2021), and evolutionary algorithms (Sefati and Navimipour, 2021), for adaptive service compositions. These approaches mainly focus on optimal service selection from available services, without paying attention to the concrete design and implementation of service compositions. Unlike these approaches, we have explored the adaptability of service compositions from the perspective of variability management (Sun et al., 2019a), in which various changes within service compositions are treated as first-class objects. Key considerations include how to identify and represent the common and variable parts in service compositions of the underlying business processes. Specifically, for positions where the service composition may change, a *variation point* is introduced, whose options are specified as *variants* representing alternatives or choices. *Dependencies* between variants at different variation points are specified as constraints. For a specific requirement, the collection of choices across all the variation points forms a *variation configuration*. In this way, various changes are realised by switching variation configurations, and thus the resulting service composition is enhanced with the adaptability.

Figure 1 shows our overall framework of variability management for adaptive service compositions (Sun et al., 2019a). The solid lines indicate the issues that have been done in previous work, while the dotted lines indicate the issues that will be addressed in this work. In our previous work, to support variability at the implementation phase (the bottom layer of Figure 1), we have designed a variability-supported service composition language named VxBPEL (Koning et al., 2009), and two versions of VxBPEL execution engines were developed (Sun et al., 2013, 2014), one is called VxBPEL_ODE (an extension to Apache ODE) and the other is called VxBPELEngine (an extension to ActiveBPEL). To support variability at the architecture design phase (the middle layer of Figure 1), we have proposed a UML

profile, called UML-VWS (Sun et al., 2010), for modelling variability in the architecture layer, and a process to manage the variability of service compositions.

Figure 1 Three lavers of variability involved in adaptive service compositions based on variability management



Since changes normally originate from requirements, they should be captured and explicitly modelled at the requirement analysis phase. Furthermore, the requirements' changes in the context of SOA may be more frequent, which implies the heavy influences on subsequent architecture design and implementation and thus the variability of service compositions at different phases should be handled in a traceable way. Consequently, a comprehensive framework should not only support the variability of service compositions throughout requirement analysis, architecture design, and implementation phases, but also retain the change traceability of software artefacts produced over these phases. However, the variability at the requirement analysis phase is not yet explicitly supported in the current framework (the top layer of Figure 1).

In this paper, we take another step towards the comprehensive framework for adaptive service compositions. We resort to variability management to systematically treat changes of business processes from the perspectives of requirements, design and implementation, and report a practical solution to achieve the adaptability of service compositions. Specifically, we focus on how to represent variable requirements and operationalise them into service compositions. Accordingly, we explore feature modelling for service compositions and model-driven techniques to further address the variability management issue at the requirements analysis phase. In particular, the feature model is used to express the requirement variation, in which common and variable requirements are explicitly captured. A set of transformation rules are designed to automatically transform a feature model to a corresponding abstract service composition model proposed in Sun et al. (2018). The main contributions of this paper are as follows:

1 A feature-driven variability-enabled adaptive service composition approach has been proposed, which extends the previous work that focused on variability design and implementation of service compositions. This work focuses on the modelling and management of adaptability requirements through a feature model and their realisation in adaptable service compositions and provides an alternative perspective to address the adaptability issue compared with the state-of-the-art work in this field.

2 A prototype has been developed to support feature modelling and model transformation of the proposed approach.

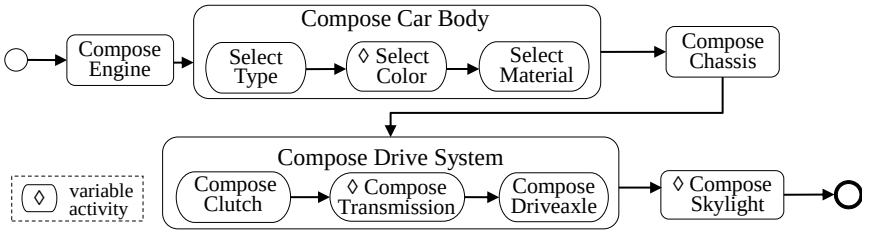
3 A case study has been conducted to validate the effectiveness of our approach and evaluate its efficiency.

The rest of the paper is organised as follows. Section 2 describes a motivating example from the car assembly domain. Section 3 introduces the underlying concepts and techniques. The proposed approach is presented in Section 4. Section 5 describes the prototype tool and Section 6 reports a case study and the evaluation results. Section 7 discusses the related work. Finally, Section 8 concludes the paper with pointing out future work.

2 Motivating example

In this section, we use a scenario from the car assembly domain (Jamie, 2022) to explain the motivation of this work.

Figure 2 Activity diagram of the car assembly process



Considering the production process of a car, each component of a car may be produced by several manufacturers in different countries, which supports the assembly need to obtain a 'complete' car. In view of software-defined everything, this assembly process involves the coordination of multiple participants, each of which can be represented by a component service. Accordingly, a simplified activity diagram of this car assembly process is shown in Figure 2. The variable requirement for some component, which has multiple alternative implementations, is marked as 'variable activity' with '◇'. The following challenges have to be responded when implementing such a composition process.

- *Challenge 1:* The requirements of users may be different from each other, varying in many aspects like colour, style, material, and size. A car manufacturer needs to provide a variety of configurations of cars to meet these varying requirements. For a complete/composed car, it must be equipped with four basic parts, i.e., 'engine', 'body', 'chassis', and 'drive system', which are common and mandatory components for a normal working car. However, the 'skylight' is not essential in general and not every user has a demand for this component. In this case, it is regarded as an optional requirement for a car. A car manufacturer needs to identify the common and variable requirements, and provides customised car composition solutions to satisfy different user requirements, while these solutions

may differ in only one or a few places. This could result in the overlap, or more precisely, redundancy, among different composition processes due to the common parts between them, causing the increase of maintenance complexity, and the decrease of production efficiency.

- *Challenge 2:* Complex constraints exist among car components in a car composition process. Take the 'body' feature for illustration, it has three mandatory attributes, namely 'body type', 'body colour', and 'body material'. For the variable requirement 'body colour', multiple colour choices are provided, such as 'grey', 'white', and 'black'. A specific colour needs to be selected for a real implementation (car) during the composition process. Obviously, there is an exclusive constraint among the colour choices. More specifically, only one colour choice is supported even though there are different alternatives. These constraints are usually hidden in user requirements, and need to be analysed and explicitly expressed in an intuitive way.
- *Challenge 3:* It is a long cycle to implement the production of a 'complete' car from user requirements to the composed process. Automated approaches or tools are needed (but are absent now) to reduce the workload of process designers and developers. To complete such a service composition, rich domain knowledge is necessary for process design and development. The composition will become increasingly complex and time-consuming because of the large amount of components involved, even to satisfy a simple user requirement. In addition, as mentioned in Section 1, inherent variability in user requirements is the root cause for such complexity in the whole process of service composition, which calls for a systematic approach to manage and keep the traceability of variations from the requirement analysis phase to the implementation phase for the car assembly process.

In this study, the above car assembly process will be used to motivate our approach since it covers all the challenges that could be faced by service compositions, regardless of which domain service compositions are used for. Moreover, the car assembly scenario has covered all the essential parts of our approach, and thus it will be also used for the case study to be reported in Section 6.

3 Background

In this section, we introduce the underlying concepts and techniques of the proposed approach, including feature-oriented domain analysis (FODA), model-driven architecture (MDA), eclipse modelling framework (EMF), and VxBPEL.

3.1 FODA

Domain analysis is a widely adopted technique for handling the variability of requirements. Thanks to domain knowledge, common and variable requirements of a specific domain can be identified. FODA (Benavides et al., 2010) is one of the most used domain analysis approaches, which identifies common and optional requirements

as features. FODA defines features as prominent or distinctive user-visible aspects, qualities, or characteristics of a software system or systems.

Features are further divided into mandatory and optional ones, which are used to represent common and variable requirements, respectively.

- *Mandatory feature* (also called *common feature*) refers to the common requirements of the domain, and is the basis of domain software artefacts. In other words, a mandatory feature is a common feature of all domain software artefacts. For a car, the basic parts such as 'engine', 'body', 'chassis' and 'drive system' are essential for all cars in the car assembly domain, which are recognised as mandatory features.
- *Optional feature* (also called *variable feature*) represents the variable requirements of the domain which is an extension of software functions and equips domain software artefacts with unique characteristics. For instance, the 'skylight' is not required for every car, and thus is an optional feature for a car.

A feature model is an abstraction of common and variable requirements of all software artefacts in a domain (Yu et al., 2016). It uses feature as the entity to express the problem space, and consists of a group of features and their relations. Feature model has been widely used to capture and represent variable requirements (Bashari et al., 2018; Narwane et al., 2016; Lian et al., 2018; Arcaini et al., 2020). In this paper, we will employ FODA for domain requirement analysis and correspondingly the feature model is adapted for modelling variable requirements of service compositions.

3.2 MDA and EMF

MDA (Sebastián et al., 2020) is an approach to software design and implementation, which provides guidelines for structuring software specifications that are expressed as models. The core activities include the model creation and transformation. In the context of service compositions, two important issues need to be considered, namely how to represent a specification model of service compositions and how to derive service composition implementations through model transformation. In this work, the specification model is represented as a feature model, while the model transformation is made of a set of the predefined model transformation rules.

EMF (Budinsky et al., 2004) is a powerful framework and code-generation facility for building Java applications based on simple model definitions. An EMF model can be defined in the form of Java interface, UML class diagram, and XML schema. The EMF model serves as a common high-level representation bridging the gap of these three forms. Modelers can define models in one form, from which others can be generated through the EMF metamodel. EMF models are represented persistently by default with XML Metadata Interchange (XMI) which is a standard interchange mechanism for serialising metadata.

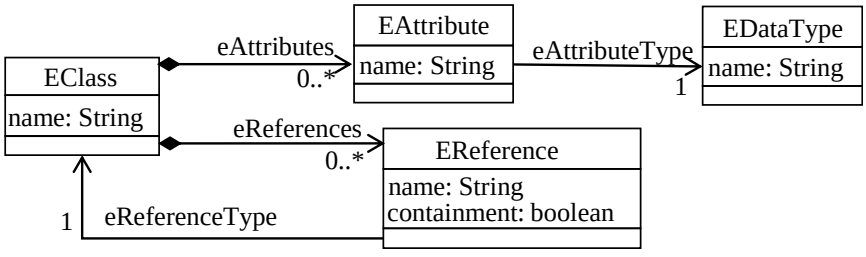
In EMF, Ecore is used to describe models, i.e., a metamodel. Figure 3 shows a simplified subset of the Ecore model, and only the core Ecore elements are introduced.

- *EClass* represents a modelled class, with a name, zero or more attributes, and zero or more references.
- *EAttribute* represents an attribute with a name and a type.

- *EReference* represents one end of an association between two classes. It includes a name, a Boolean flag to indicate if it represents a containment, and a reference type which is another class.
- *EDataType* represents the type of an attribute like int, float, or java.util.Date.

Due to advantages such as popularity and interchangeability, we employ EMF to represent the feature model and the service composition implementation of the proposed approach, and the feature modelling prototype is also based on EMF. Furthermore, we turn to EMF-based model transformation for enabling the traceability of changes between different phases of service composition development. In this way, the proposed approach reduces not only the workload of service composition development, but also the complexity of service composition artefacts.

Figure 3 Simplified subset of the Ecore model



3.3 VxBPEL

VxBPEL (Koning et al., 2009) is an extension of the standard WS-BPEL to support variability of service compositions. It provides variability constructs like *variation point*, *variant* and *realisation relations*. To be consistent with WS-BPEL, these newly added constructs are represented using an XML-based form and used to implement variable business processes in a declarative way. The syntax of each VxBPEL activity/element is illustrated below.

- <*VariationPoint*> represents the position possible to change. It contains multiple available operations, and each substitute is expressed with a <*Variant*> element.
- <*Variants*> consists of a group of <*Variant*> which represents a scenario constructed by a group of activities. It is defined with a function or operation to describe an alternative plan. The contents of <*Variant*> can be WS-BPEL codes, or <*VariantPoint*> elements for more complex variability design. By default, there is an exclusive constraint among the alternative variants.
- <*VariabilityConfigurationInformation*> refers to the variability configurations of the process.
- <*ConfigurableVariationPoints*> consists of a group of <*ConfigurableVariationPoint*> which refers to the collection of all configurations, with a unique id and a default configuration identified by *defaultVariant*.

- <*Rationale*> is the text description for the element of <*ConfigurableVariationPoint*>.
- <*VariantInfo*> is the text description for <*Variant*>.
- <*RequiredConfiguration*> refers to a specific configuration.
- <*VPChoices*> consists of a group of <*VPChoice*> which refers to the configurations of variants at the variation points.
- <*Constraint*> refers to the constraints between variants, with three attributes involving *CType*, *variantS* and *variantT*. *CType* refers to the type of constraint, including *Inclusion*, *Exclusive*, *Exclusion*, *Substitution* and *Corequisite* (Sun et al., 2019a). *variantS* and *variantT* refer to the source and target of a constraint, respectively.

Based on VxBPEL, we have further proposed a variation-based abstract service composition model (Sun et al., 2018). The model is first used to express the business process logic, where positions having possible changes are abstracted as variation points, and possible implementations are abstracted as variants within the variation points. The execution engine VxBPEL_ODE (Sun et al., 2014) is then used to configure the abstract service composition model according to user configuration files, and finally derive the business process instances at run-time to meet different user requirements. The multiple process instances are isolated from each other and can co-exist at the same time.

4 Approach

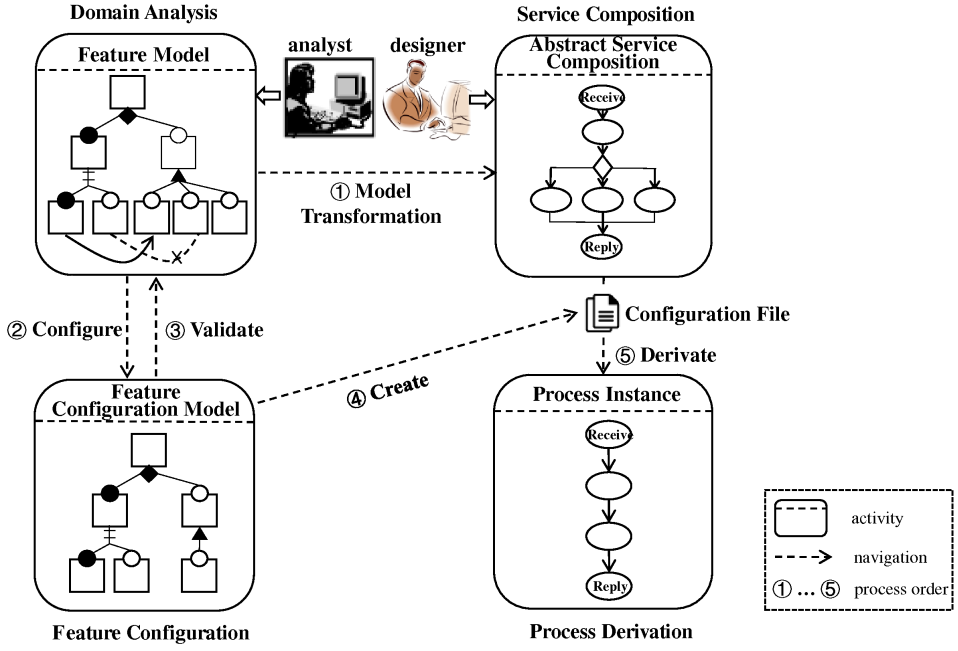
In this section, we first give an overview of the proposed approach, then examine its key issues including feature model, abstract service composition model, and transformation between the feature model and the abstract service composition model, and finally discuss the feature configuration and verification.

4.1 Approach overview

To address the challenges mentioned in Section 1, we propose a feature-driven variability-enabled adaptive service composition approach, as shown in Figure 4. Here, the marked numbers represent the order of interactions between activities, and the arrows represent the navigation between activities, namely from the source activity to the target activity. In our previous work, service compositions with variability are designed without taking requirements variability into account, although they can be customised to satisfy different user requirements. This work focuses on addressing the two key issues, namely how to explicitly model the requirements with variability of service compositions and how to transform such a specification model to service composition implementation. As a result, the proposed approach together with the previous framework constitutes a comprehensive framework that supports variability management cross the full life-cycle of service compositions, from requirements analysis, design, implementation, to execution and maintenance. The proposed approach has the following major components.

1 *Domain analysis:* The main task of this component is to identify commonalities and variations of requirements in terms of domain analysis. To do that, domain knowledge is crucial for obtaining common and variable requirements as well as constraints among these requirements. Specifically, a feature model is presented for representing domain requirements in the format of XMI schema, and the subsequent feature selections are automatically mapped to variation configurations of the business process, resulting in automatic derivation of user-specific process instances. More details on feature modelling will be discussed in Subsection 4.2.

Figure 4 Overview of the approach (see online version for colours)



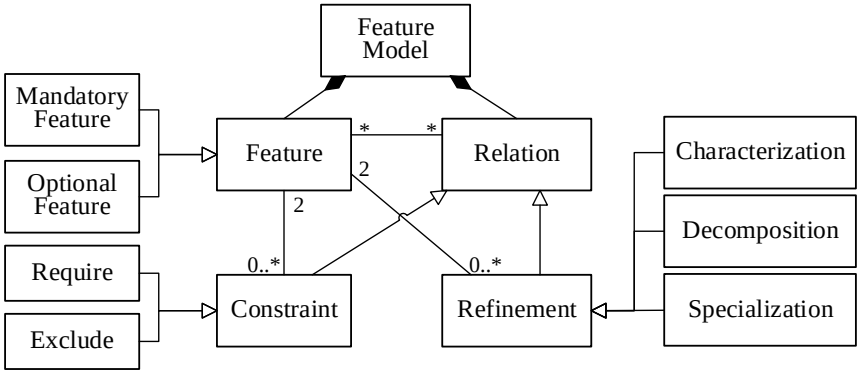
2 *Service composition:* To cater for both common and variable requirements in an application domain, variable business processes are required. This can be done by defining rules and algorithms that determine the mapping of elements in the feature model to the element activities in the abstract service composition model. Fortunately, performing such a model transformation becomes easy, because the Ecore model as discussed in Subsection 3.2 can represent both the feature model and the abstract service composition model. What we need to do is to specify the metamodel of both models and the relationships between their model elements. More details on the representation of the feature model and the abstract service composition model using the Ecore model will be discussed in Subsections 4.2.2 and 4.3, respectively, while model transformation will be discussed in Subsection 4.4.

3 *Feature configuration:* The domain requirements expressed as a feature model contain both mandatory and optional ones. For a specific or personalised requirement, the feature model needs to be customised by specifying (or selecting)

the optional features, resulting in a feature configuration model. Such a configuration model will be automatically verified to determine whether or not it conforms to the predefined constraints among features. Once the feature model is properly configured, the feature configuration will be used to guide variant selection of the abstract service composition model, and a user configuration file is generated for subsequent process derivation. More details on feature configuration and verification will be discussed in Subsection 4.5.

4 *Process derivation:* The abstract service composition model derived from the feature model through the automatic model transformation contains only the skeleton of a business process, and is not an executable business process. This service composition model is represented as a VxBPEL-based specification, which has to be further customised according to the user configuration file corresponding to a feature configuration model. During the customisation, necessary implementation details [using the VxBPEL Designer (Sun et al., 2019a)] will be enriched in order to form a complete and executable business process. More details on process derivation will be discussed in Subsection 4.6.

Figure 5 Metamodel of the proposed feature model



Compared with the related works in this area, the proposed approach has the following new features:

1 The variability of requirements for service compositions is explicitly treated, which is different from those approaches related to BPMN (Mazzara et al., 2012; Terenciani et al., 2015), WS-BPEL extension (Karastoyanova and Leymann, 2009; Cherif et al., 2015) and UML (Sun et al., 2010; He et al., 2015) that mainly focus on variability design or implementation without considering variability of requirements. In our approach, variable requirements are represented in a more intuitive and concrete way with the feature model, which makes it easier for users and process designers to understand and manage the variability of requirements.

2 The proposed approach advances related works based on feature modelling and configuration (Nguyen et al., 2016; Alférez et al., 2014) that only focus on variable requirements without considering variability design and implementation. In our approach, the variability of requirements is automatically mapped onto that

of the abstract business process through model transformation, which not only greatly reduces the efforts required for service composition implementation, but also achieves the traceability of variability during the full life-cycle of service compositions.

4.2 Feature model

The primary issue we have to address is how to explicitly model the variable requirements of service compositions. Inspired by the FODA approach, we adopt the feature model as a base model for requirements analysis in our approach. We first give the definition of the feature model, then present its syntax in Backus-Naur form (BNF), finally discuss the construction of its Ecore model using EMF.

Table 1 Definition of constraints

Constraint	*Description*	*Denotation*
Require	If feature X is selected then feature Y must be selected	$X \rightarrow Y$
Exclude	Feature X and feature Y should not be selected simultaneously	$X \nrightarrow Y$

Table 2 Definition of refinements

Refinement	*Description*	*Parent-role*	*Child-role*
Decomposition	Refine a feature into its constituent features	*Whole*	*Part*
Characterisation	Refine a feature by identifying its attribute features	*Entity*	*Attribute*
Specialisation	Refine a general feature into a feature incorporating further details	*General-entity*	*Specialised-entity*

Table 3 Guidelines for feature modelling

No.	*Description*
G_1	The top-level feature (the feature with the highest level of abstraction) should be a mandatory feature.
G_2	Only one type of refinements can be defined between each feature and its child features.
G_3	At least one of the child features is mandatory if there is a decomposition refinement between the parent feature and its child features.
G_4	The child feature can be either mandatory or optional if there is a characterisation refinement between its parent feature and itself.
G_5	All child features are optional if there is a specialisation refinement between their parent feature and themselves.
G_6	Either characterisation or specialisation, instead of decomposition, is allowed between a feature and its child features if there is a characterisation refinement between this feature and its parent feature.
G_7	A feature should not be further refined if there is a specialisation refinement between the feature and its parent feature.
G_8	Constraints should only be defined between optional features.

4.2.1 Definition of the feature model

Feature modelling is a popular requirements analysis technique (Benavides et al., 2010). To support the feature modelling of service compositions, we design a feature model in terms of a metamodel which provides a standard and interchangeable representation and reflects fundamental characteristics of domain analysis. The metamodel of the proposed feature model in the UML class diagram is shown in Figure 5. We next introduce elements, relations between elements, and their constraints in the model.

Figure 6 Syntax of the feature model

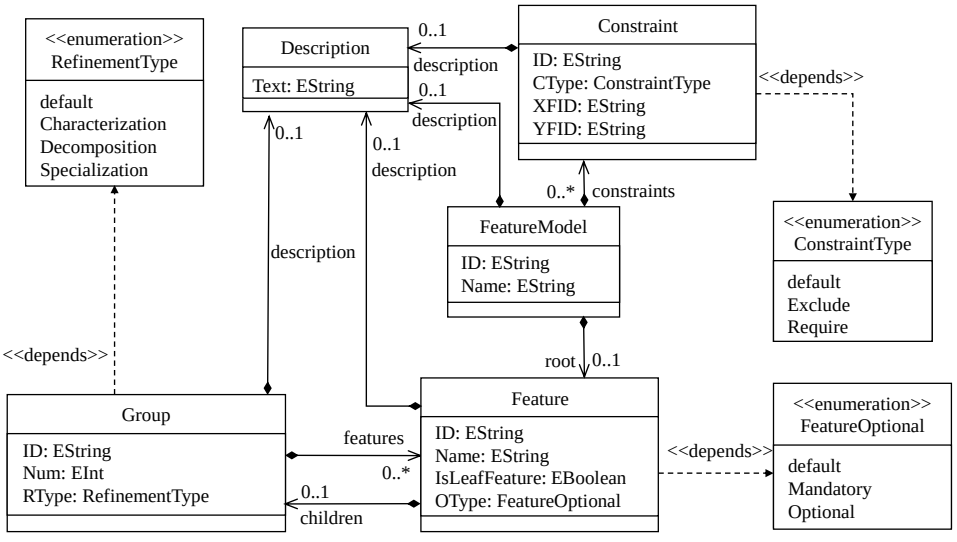
```
FeatureModel ::= <ID><Name><RootFeature><Constraints>[Description]
RootFeature ::= [Group]<ID><Name><IsLeafFeature>
                <FeatureOptional>[Description]
Group ::= {Feature}<ID><Num><RefinetmentType>[Description]
Feature ::= [Group]<ID><Name><IsLeafFeature>
            <FeatureOptional>[Description]
IsLeafFeature ::= True|False
FeatureOptional ::= Mandatory|Optional
RefinementType ::= Decomposition|Characterization|Specialization
Constraints ::= <ID><ConstraintType><XFID><YFID>
ConstraintType ::= Require|Exclude
XFID ::= <ID>
YFID ::= <ID>
Description ::= <Text>
Text ::= {a|…|z|A|…|Z}
ID ::= {a|…|z|A|…|Z}
Name ::= {a|…|z|A|…|Z}
Num ::= < -1|0|1|2|…|N>
```

A feature model is identified by the *feature model* element, which consists of a group of features and relations among them, and the two are aggregated through the *feature model* class. *Feature* is specialised into *mandatory feature* and *optional feature*, which indicate common and optional features/requirements, respectively. *Relation* is specialised into *constraint* and *refinement* according to the levels of feature abstraction. *Constraint* defines the limitations between features at the same levels of abstraction, and aims to enhance the control capability for domain requirements. According to our previous work (Sun et al., 2019a), two types of constraints are defined, named as *require* and *exclude*, and their semantics are explained in Table 1. *Refinement* integrates features at different levels of abstraction into a hierarchical structure. As explained in Table 2, there are three types of refinement, including *decomposition*, *characterisation* and *specialisation*. To a large extent, more complex relations can be reduced to the combinations of these relations. A feature at a high abstraction level is called *parent feature*, and a feature refining a parent feature is called *child feature*. Furthermore, a set of guidelines as shown in Table 3 are provided to further guide the modelling process. These guidelines must be followed during the construction of feature models in order to ensure the correctness and consistency and avoid the potential misuse due to lack of domain knowledge.

With the feature model discussed above, one can describe the domain requirements in an accurate and formal way. All elements and the relationships between them are

defined with clear semantics, and the guidelines are also provided to guide the feature modelling procedure and ensure the consistency of the resulting feature model.

Figure 7 Representation of the feature model using Ecore



4.2.2 BNF and Ecore model of feature model

We now present the syntax of the feature model using the BNF (Backus et al., 1963). Figure 6 provides a formal definition of the proposed feature model, in which the *Root* feature refers to the top-level feature, and the *leaf* feature refers to a feature without any child features.

Based on the above metamodel, the Ecore model of EMF is further employed to represent the feature model, as shown in Figure 7. The *EClass* concerned includes *FeatureModel*, *Feature*, *Group*, *Description* and *Constraint*. The *EEnum* includes *FeatureOptional*, *RefinementType* and *ConstraintType*. The key *EClass* elements are introduced below.

- *FeatureModel* is a high level abstraction of the domain, with *EAttributes ID* and *Name*. *ID* is the unique identifier of FeatureModel, and *Name* shows the meaning of *EClass*. *Description*, *Feature* and *Constraint* are aggregated through *FeatureModel*, indicating the *FeatureModel* with 0 or 1 *description*, 0 or 1 *root*, and 0 or more *constraints*, respectively.
- *Feature* is the abstraction of a feature. *EAttribute IsLeafFeature* indicates if the feature is a *leaf* feature. EEnum *OType* shows the feature type, mandatory or optional. Aggregation *children* indicates the *Feature* with 0 or 1 *Group*.
- *Group* represents the hierarchy of the feature model. The value of *EAttribute Num* is –1 if all features included in the *Group* are *Leaf* features; otherwise, it counts the number of *Feature* in the *Group*. *EEnum RType* indicates the refinements between the feature in the *Group* and its *Parent* feature. Aggregation *Features* indicates the *Group* with 0 or 1 *Feature*.

- *Constraint* represents the constraint in the feature model. The constraint type is defined with *EAttribute CType*. *EAttributes XFID* and *YFID* are the two features bound to a constraint.
- *Description* is the illustration for *EClass* and *EEnum*. *EAttribute Text* is the text content of the description.

4.3 Abstract service composition model

To realise both common and variable requirements, the abstract service composition model described in Subsection 3.3 is used. In this service composition model, the common business logic is handled by the traditional WS-BPEL, while the possible changes of the business process are left to the variability constructs provided by VxBPEL. Since WS-BPEL has been well studied, we will focus on the syntax of VxBPEL and its representation using the Ecore model, which is the basis for the subsequent model transformation as discussed in Subsection 4.4.

Figure 8 Syntax of VxBPEL

VxBPEL	::=	<Process>{WSDL}
Process	::=	<Name of Process>{variable}{PartnerLink}{faultHandle}{eventHandle}{activity}{Constraints}
PartnerLink	::=	<Name of PartnerLink>[myrole | partnerrole]
myrole	::=	<Name of role>
partnerrole	::=	<Name of partnerrole>
variable	::=	<Name of Variable><Type Name of Message>
activity	::=	<atomic activity | structured activity>
atomic activity	::=	<invoke | receive | reply | empty | assign | wait | throw>
structured activity	::=	<sequence | flow | if | pick | while | VariationPoint |Variants | Variant | VariabilityConfigurationInformation | ConfigurableVariationPoints | ConfigurableVariationPoint | Rationale | VariantInfo | RequiredConfiguration | VPChoices | VPChoice | Constraint>
VariationPoint	::=	<Variants><Name of VariationPoint>{activity}
Variants	::=	<Variant>
Variant	::=	[VariantInfo][RequiredConfiguration]<Name of Variant>{activity}
VariabilityConfigurationInformation	::=	<ConfigurableVariationPoints><Name of VariabilityConfigurationInformation>
ConfigurableVariationPoints	::=	<ConfigurableVariationPoint>
ConfigurableVariationPoint	::=	<Variants>[Rationale]
RequiredConfiguration	::=	<VPChoices>
VPChoices	::=	{VPChoice}
VPChoice	::=	<vpname><variant>
vpname	::=	<Name of VariationPoint>
variant	::=	<Name of Variant>
Constraints	::=	<ConstraintType><variantS><variantT>
ConstraintType	::=	<Inclusion | Exclusive | Exclusion | Substitution | Corequisite>
variantS	::=	<Name of Variant>
variantT	::=	<Name of Variant>
Rationale	::=	<Text>
VariantInfo	::=	<Text>
Text	::=	{a|…|z|A|…|Z}

The formal of syntax of VxBPEL using BNF is shown in Figure 8. Similar to the feature model, the abstract service composition model can also be represented in the Ecore model of EMF. Since VxBPEL is an extension of WS-BPEL with variability constructs, the Ecore model of VxBPEL consists of two parts: one for WS-BPEL (refer to https://eclipse.googlesource.com/bpel/org.eclipse.bpel/+/MultiTab-DomFacade/plugins/org.eclipse.bpel.model/src/model/bpel.ecore), as shown in Figure 9, and the other for variability constructs provided in VxBPEL, as shown in Figure 10.

Note that Figure 9 only shows a subset of the Ecore model of WS-BPEL for simplicity, while Figure 10 mainly shows the Ecore model for variability constructs.

Specifically, the *Activity* class is extended for the construction of *VariationPoint*, *Variants*, *Variant*, *VPChoices*, and *VPChoice*, and the *BPELExtensibleElement* is extended for the construction of *VariabilityConfigurationInformation*, *ConfigurableVariationPoints*, *ConfigurableVariationPoint*, *RequiredConfiguration*, and *Constraint*. The *EEnum ConstraintType* supports the five types of variant dependencies, and the classes *VariantInfo* and *Rationale* are supported by extending the *Documentation* class.

Figure 9 Fragment of the Ecore model of WS-BPEL

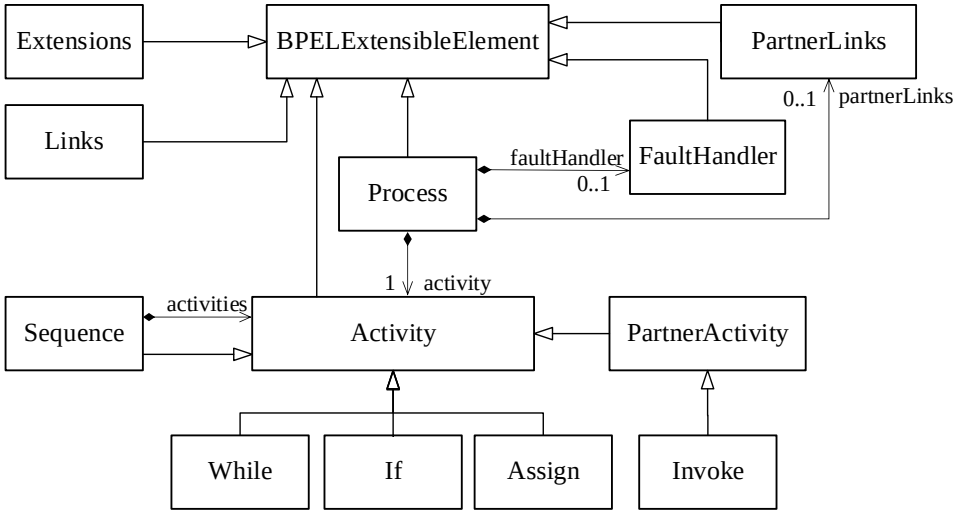
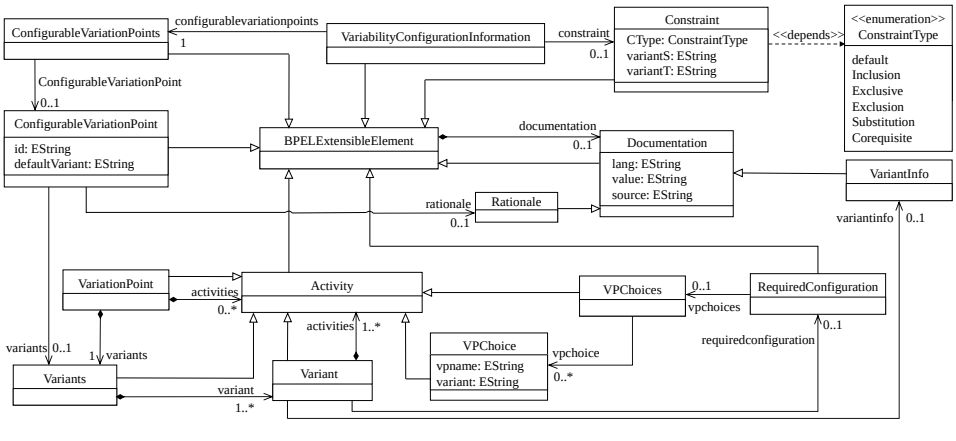


Figure 10 Ecore model of the extended constructs of VxBPEL



4.4 Model transformation

As mentioned in Subsection 4.1, model transformation will be used to reduce the effort required by service composition development and trace the variability at different phases. In our case, the feature model representing mandatory and optional requirements

serves as the source model, while the abstract service composition model with variability design is the target model. Before model transformation, a clear understanding of the syntax and semantics of both the source and target models is necessary, as formally defined in Subsections 4.2.2 and 4.3, respectively.

Among various techniques available (Czarnecki and Helsen, 2006), we adopt metamodel mapping for the implementation of model transformation since both metamodels of source and target models are represented using the Ecore model. Furthermore, the key is to design the mapping rules for key elements between source and target metamodels, and these mapping rules will then guide the transformation process to produce the target model.

4.4.1 Mapping rules

Typically, SOA adopts a vertical service composition style in which a component service normally implements a specific functionality (or feature) corresponding to an activity in the resulting service composition. In this context, a natural way of designing mapping rules is to map features of the feature model to activities of the abstract service composition model. Furthermore, each feature in the source model should be mapped to a corresponding activity in the target model. Taking into account the semantics and syntax of source and target models, we design the following mapping rules:

- *Rule 1:* A *mandatory leaf feature* of the source model is mapped to an *invoke* activity of the target model.
- *Rule 2:* An *optional leaf feature* of the source model is mapped to the variability design in the target model, i.e., mapping this feature to a *variation point* and a *variant* of the target model.
- *Rule 3:* An *optional feature*, which is neither a *leaf feature* nor holding a *specialisation* refinement with its child features, is mapped to the variability design in the target model, i.e., mapping this feature to a *variation point* and a *variant* of the target model.
- *Rule 4:* A non-leaf feature which holds a *specialisation* refinement with its child features is mapped to the variability design in the target model, i.e., mapping this feature to a *variation point* and its child features as *variants*.

4.4.2 Transformation algorithms

With the mapping rules above, we further design transformation algorithms to automate the transformation process. Algorithm 1 describes how a feature model is transformed to a VxBPEL-based abstract service composition model at a high level, the input is a feature model (i.e., the source model), and the output is an abstract service composition model (i.e., the target model).

Algorithm 2 implements the transformation process in an iterative procedure, namely repeatedly mapping a feature of the feature model to an activity of the abstract service composition model according to the applicable mapping rules. We further explain the transformation procedure of Algorithm 2 as follows:

Algorithm 1 Model transformation

Require: A feature model fm;
Ensure: An abstract service composition model vp.
1: **procedure** Convert(fm, vp)
2: Initialise vp, process $\leftarrow \varnothing$, WSDL $\leftarrow \varnothing$;
3: Define Feature f = null, Activity $a \leftarrow$ *Sequence* Activity;
4: Get Root Feature by parsing fm;
5: $f \leftarrow$ Root Feature;
6: Convert f into $process$ activity;
7: Add a under $process$ activity;
8: HandleFeature(f, a); ◁ **Algorithm 2**
9: **end procedure**

Algorithm 2 Feature mapping

Require: A feature of the feature model f;
Ensure: An activity of the abstract service composition model asc.
1: **procedure** HandleFeature(f, asc)
2: Define Refinement Type $rtype$ = null, Feature f_child = null, Activity a_vp = null;
3: **if** f.isLeafFeature == 'true' **then**
4: **if** f.OType == 'Mandatory' **then**
5: Convert f into *invoke* activity under asc; ◁ **Rule 1**
6: **end if**
7: **if** f.OType == 'Optional' **then**
8: Convert f into $VariationPoint$ activity and $Variant$ activity under asc; ◁ **Rule 2**
9: **end if**
10: **else**
11: $rtype \leftarrow$ refinement type between f and f's child features;
12: **if** f.OType == 'Mandatory' and $rtype \neq$ 'Specialisation' **then**
13: **for** each child feature of f **do**
14: $f\_child \leftarrow$ child feature of f;
15: HandleFeature(f_child, asc); ◁ **handling child features**
16: **end for**
17: **end if**
18: **if** f.OType == 'Optional' and $rtype \neq$ 'Specialisation' **then**
19: Convert f into $VariationPoint$ activity and $Variant$ activity under asc; ◁ **Rule 3**
20: $a\_vp \leftarrow Variant$ activity;
21: **for** each child feature of f **do**
22: $f\_child \leftarrow$ child feature of f;
23: HandleFeature(f_child, a_vp);
24: **end for**
25: **end if**
26: **if** $rtype$ == 'Specialisation' **then**
27: Convert f into $VariationPoint$ activity under asc; ◁ **Rule 4**
28: **for** each child feature of f **do**
29: $f\_child \leftarrow$ child feature of f;
30: Convert f_child into $Variant$ activity;
31: **end for**
32: **end if**
33: **end if**
34: **end procedure**

Step 1 For a given feature of the feature model, it is necessary to determine whether this feature is a leaf one or not. If yes (line 3), jump to *step 2*; otherwise (line 10), jump to *step 3*.

Step 2 Determine the type of the feature. If the feature is a mandatory one, it is mapped to an *invoke* activity of the abstract service composition model according to *rule 1* (lines 4–6); if the feature is an optional one, it is mapped to a *variation point* and also a *variant* according to *rule 2* (lines 7–9).

Step 3 Determine both the type and refinement relation between this feature and its child features. If the feature is an optional one and the refinement is not specification, it is set as a *variation point* and a *variant* according to *rule 3* (lines 18–25), and then recursively back to *step 1* to handle each child feature of this feature; if the refinement is specification, it is mapped to a *variation point*, and its child features are mapped to *variants* under the *variation point* according to *rule 4* (lines 26–32); otherwise (lines 12–17), recursively back to *step 1* to handle each child feature of this feature.

4.5 Feature configuration and verification

Given a feature model corresponding to requirements of all service compositions of a domain, the requirements for a specific service composition can be met through a feature configuration. Specifically, since the feature model contains both mandatory and optional features, a configuration model can be obtained through the customisation, i.e., by activating or deactivating optional features. This configuration process must follow the constraints between features. As defined in Figure 6, a constraint is a four-tuple *<ID, ConstraintType, XFID, YFID>*, where (*ID*) indicates the identifier of the constraint, (*ConstraintType*) refers to the constraint relation, (*XFID*) and (*YFID*) refer to source and target feature, respectively. Since we have formally defined the semantics for various types of constraints, the verification of configuration model is automated. Only when all the feature constraints are satisfied, is a feature configuration valid. Finally, a user configuration file is produced, which will be used to drive the derivation of the business process instance.

4.6 Configuration-based process derivation

Note that the target process derived by the above model transformation is just a skeleton of service composition framework consisting of a set of key activities and their hierarchy (i.e., sequence, parallel or variation relationships). More details have to be added, such as binding concrete services to the *invoke* activities, adding *link* activities in the desired order, and adding the namespace of involved services. These tasks become easy with the aid of VxBPEL Designer. At this phase, the execution engine, such as VxBPEL_ODE or VxBPELEngine, can be used to deploy the VxBPEL specification and derive the process instance according to the given user configuration file. More details on process derivation based on the user configuration file can be found in Sun et al. (2018).

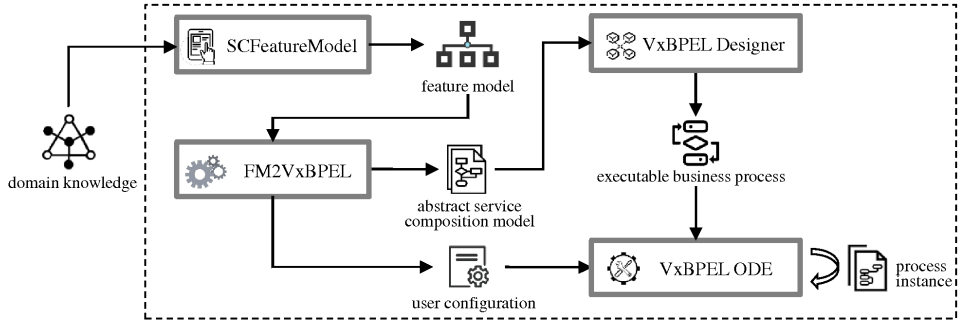
5 Prototype tool

A prototype tool has been developed to further support the proposed approach. There are two main components, one for feature modelling called SCFeatureModel and the other for feature-driven service composition called FM2VxBPEL. We first give an overview of the prototype, and then discuss features of the two components, respectively.

5.1 Overview of the prototype

Figure 11 shows an overview of the prototype. The components SCFeatureModel and VxBPEL Designer (Sun et al., 2019a) are implemented as plug-ins of Eclipse, whereas the components FM2VxBPEL and VxBPEL_ODE (Sun et al., 2014) are implemented as stand-alone Java applications. The SCFeatureModel provides visualisation for feature modelling, and the constructed feature model is persistently stored in an XMI file. The FM2VxBPEL is used to perform the model transformation as well as the feature customisation based on the feature configuration. The target abstract service composition model needs to be further completed with the VxBPEL Designer to obtain the executable variability-based business process. At run-time, the execution engine VxBPEL_ODE will derive the specific process instance according to the user configuration of the feature model. We further examine SCFeatureModel and FM2VxBPEL in the following subsections.

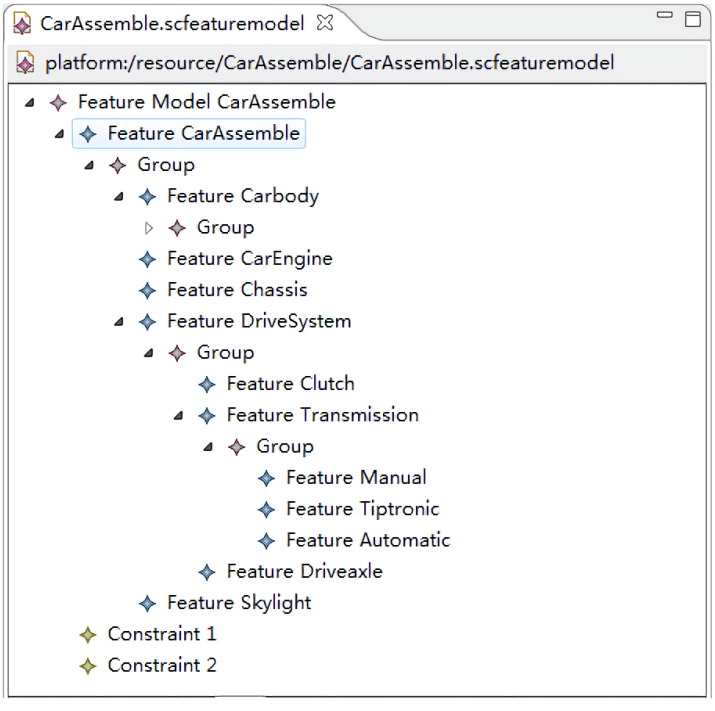
Figure 11 Overview of the prototype



5.2 SCFeatureModel

The SCFeatureModel is used to construct the feature model. The domain analyst first analyses the common and variable requirements and their relations in a domain, and then use the SCFeatureModel to establish the feature model instance for a specific domain. Figure 12 illustrates the feature model and its constraints for the car assembly domain with a screenshot of the SCFeatureModel component. The feature model is persistently stored following the XMI schema which can be viewed through this component.

Figure 12 Feature model of car assembly using SCFeatureModel (see online version for colours)



5.3 FM2VxBPEL

The main functionality of FM2VxBPEL is model transformation. In addition, the component also supports visual customisation of feature models, generation of user configuration files, and deployment and execution of business process instances. FM2VxBPEL consists of five modules containing feature model management, requirements configuration, business process management, business process execution, and execution engine. Each of the modules is discussed below.

1 *Feature model management* is responsible for the management of the feature model. It has the following three components:

- *Feature model visualisation:* The XMI file of the feature model is parsed using dom4j into a feature model diagram for visualised representation.
- *Configurable feature tree construction:* The feature model tree is constructed by parsing the XMI file of the feature model and organising the features with the JTree technique. It is configurable by the users with the aim to meet diverse user requirements.
- *Model transformation:* This component is the core of the FM2VxBPEL, which can transform the feature model to the abstract service composition model following the predefined transformation rules.

2 *Requirements configuration* supports the customisation of the feature model according to the requirements of different users. It has the following three components:

- *Visualised feature configuration:* By configuring the optional features of the feature model, users can customise the feature model in accordance with their specific requirements.
- *User configuration verification:* The user configuration is automatically verified to determine whether or not it satisfies the constraints between the features. The user configuration file is created only when all constraints defined are satisfied.
- *User configuration file creation:* The user configuration file contains the configuration of variation points and variants corresponding to the configuration of feature model. It is stored in the XMI format.

3 *Business process management* is responsible for the management of business processes. It has the following four components:

- *Deployment and undeployment:* This component is implemented by reusing the process management interface provided by Apache ODE to support the deployment and undeployment of business processes.
- *Operation log query:* The process manager can look up the deployment and execution log.
- *Process tree creation:* This component parses the business process file and creates the process tree.
- *Process instances derivation:* At run-time, the engine VxBPEL_ODE will parse the user configuration file, and derive the process instance based on the configuration of variation points and variants in the user configuration file.

4 *Business process execution* is a simulation of a SOAP client. A SOAP message template is created by parsing the WSDL file using dom4j to get the address, namespace, operation names and request messages of the service. It can create the SOAP request message based on the request parameters provided by users and return the execution results to users.

5 *Execution engine* provides an interface for the startup and shutdown of the execution engine VxBPEL_ODE.

6 Case study

In this section, we report on a case study where the car assembly scenario introduced in Section 2 is used for evaluation. There are threefold reasons for this arrangement:

1 It is a typical scenario manifesting variable requirements, which is well investigated and thus easily understood.

2 It normally employs product family techniques for design and thus domain analysis applies.

3 The WS-BPEL service composition implementation is available in previous studies, which saves the effort required for the experimental evaluation in this study.

Through the case study, we hope to answer the following questions:

1 Can the proposed approach cover typical types of requirement variability?

2 Can the proposed approach respond to varying requirements easily and efficiently?

3 What is the performance cost of the proposed approach?

To answer the first two questions, we demonstrate the application of the proposed approach and report our observations, while for the third question, we evaluate the time cost for variability-related processing and the execution of business process. Next, we first report the application of the proposed approach and observations, then evaluate the performance cost, and finally conclude the case study with some further remarks.

6.1 Application of the proposed approach and observations

We apply the proposed approach to the car assembly scenario and demonstrate the main artefacts in a step-wise manner as follows.

6.1.1 Feature model

Domain requirements for the car assembly scenario have been analysed in Section 2. For demonstration purpose, we assume the following constraints in this scenario:

a If the 'white colour' for 'body' is chosen, the 'manual' transmission system must be chosen at the same time.

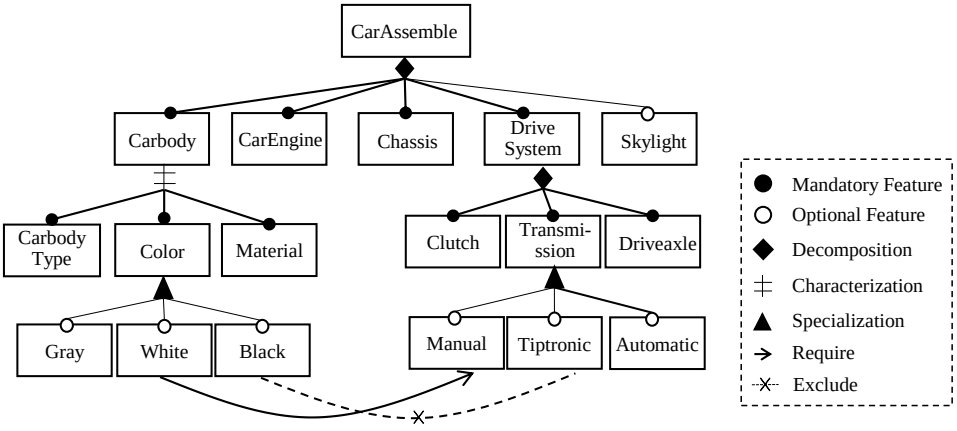
b If the 'black colour' for 'body' is chosen, the 'tiptronic' transmission system cannot be chosen.

The SCFeatureModel component of the prototype presented in Subsection 5.2 is employed to model the features of this scenario, and the resulting feature model, as shown in Figure 13, represents commonalities and variations of requirements as well as their relations.

The feature model provides an intuitive representation for the requirements of the car assembly system. 'Carbody', 'CarEngine', 'Chassis', and 'DriveSystem' are common and essential for a car, and thus expressed as mandatory features. 'Skylight' is an optional feature that can be customised by users. The 'DriveSystem' is decomposed into three sub-features, i.e., 'Cluth', 'Transmission', and 'Driveaxle'. 'Carbody' has a characterisation relation with its child features, and all its child features are mandatory. 'Colour' can be specialised by any of the features 'grey', 'white', or 'black', while 'transmission' is specialised by any of 'manual', 'tiptronic', or 'automatic'. Moreover, there exist two groups of constraints corresponding to our assumptions. The *require*

constraint means that the ‘manual’ feature must be chosen in case the ‘white’ feature is chosen. The simultaneous selection of the ‘black’ and ‘tiptronic’ feature is not permitted due to the *exclude* constraint between these two features. In addition, the *exclude* constraints exist mutually between the optional sub-features of the ‘colour’ feature and the ‘transmission’ feature, respectively, while these constraints are omitted for simplicity in Figure 13.

Figure 13 Feature model of the car assembly system



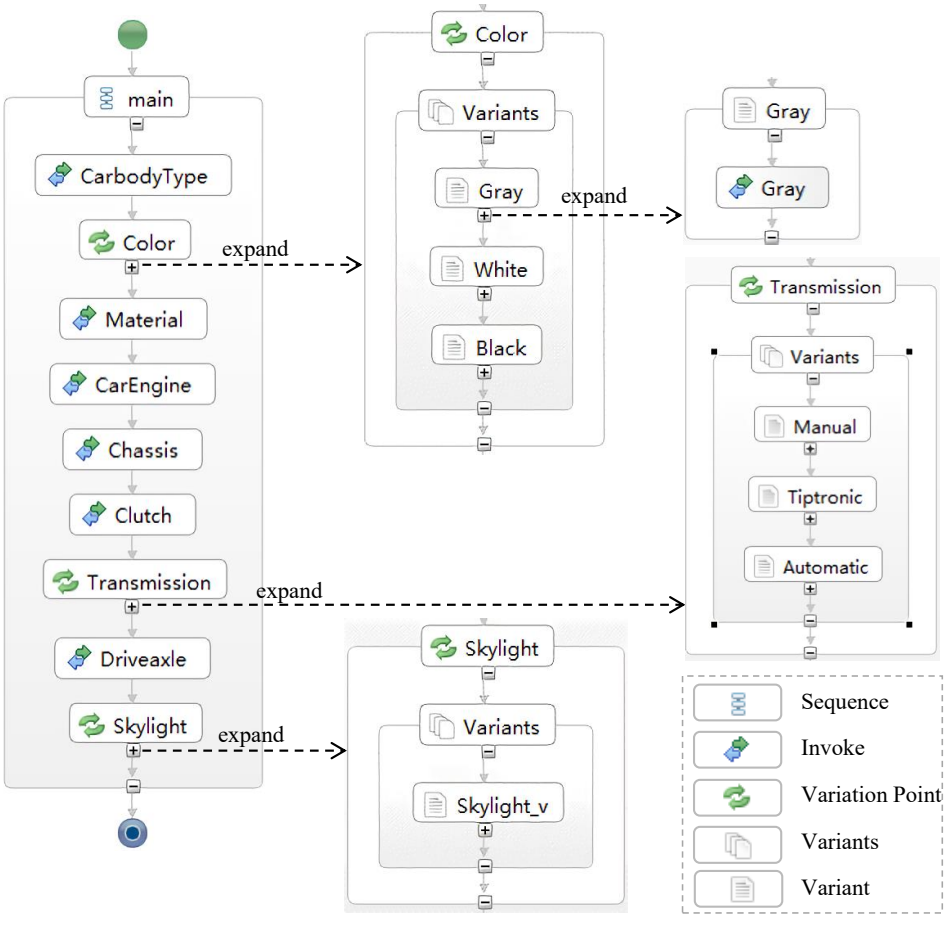
6.1.2 Abstract service composition

The feature model is then used for deriving an abstract service composition according to the transformation rules. The FM2VxBPEL component of the prototype presented in Subsection 5.3 is employed to perform the transformation process. Specifically, ‘CarEngine’, ‘Chassis’ and other mandatory leaf features are mapped onto invoke activities. ‘Colour’ is mapped onto a variation point, and the three optional colour schemes (i.e., ‘grey’, ‘white’ and ‘black’) are mapped onto variants under the variation point ‘colour’. In addition, the optional feature ‘skylight’ is mapped onto a variation point, and only one variant named ‘Skylight_v’ is associated.

The resulting abstract service composition model is a skeleton of the business process framework, which mainly consists of necessary activities such as sequence, invoke and variation activities. We further employ VxBPEL Designer to add implementation details. Accordingly, the flowchart of the executable variable business process is shown in Figure 14. For instance, there are three variants for the variation point ‘colour’, namely ‘grey’, ‘white’, and ‘black’, and its implementation based on VxBPEL is demonstrated in Figure 15.

From the artefacts illustrated above, one can observe that the traceability of variations at the phase of requirement analysis and service composition implementation is well supported due to the transformation process. On one hand, each feature in the feature model corresponding to domain requirements is transformed to an activity in the resulting business process. On the other hand, the commonalities and variations of requirements are also reflected by variability constructs in the resulting business process.

Figure 14 Flowchart of variable business process of the car assembly (see online version for colours)

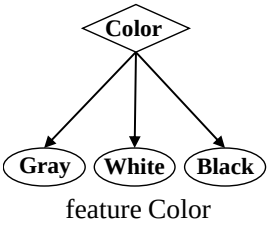


6.1.3 Feature configuration model

The feature configuration is used to represent varying user preferences for a car composition, which is done with the aid of FM2VxBPEL. The feature configuration is customised by the activation of optional features.

As an illustration, we assume that the user wants to compose a car with the white colour, a manual transmission, and a skylight. In this context, the 'white', 'manual' and 'skylight' features are selected in the feature configuration. Then, a verification process happens before such a feature configuration is activated. Obviously, this feature configuration is valid since it conforms to the *require* and *exclude* constraints in the car assembly. Next, the feature configuration is used to guide the variation configuration of the abstract service composition, namely 'white', 'manual' and 'Skylight_v' variants are to be selected together in the expected business process. Accordingly, a user configuration file is produced as shown in Figure 16.

Figure 15 VxBPEL code segment of the variation point 'colour'



VxBPEL Code

```
<vxbpel:VariationPoint name="Color">
 <vxbpel:Variants>
  <vxbpel:Variant name="Gray">
   <bpel:sequence name="Color_Gray">
     <invoke name="Gray"></invoke>
     ...
   </bpel:sequence>
 </vxbpel:Variant>

  <vxbpel:Variant name="White">
   <bpel:sequence name="Color_White">
     <invoke name="White"></invoke>
     ...
   </bpel:sequence>
  </vxbpel:Variant>

  <vxbpel:Variant name="Black">
   <bpel:sequence name="Color_Black">
     <invoke name="Black"></invoke>
     ...
   </bpel:sequence>
  </vxbpel:Variant>
 </vxbpel:Variants>
</vxbpel:VariationPoint>
```

Figure 16 Illustration of the user configuration file

```
<UserConfig name="default" xmlns:vxbpel="http://vxbpel.rug.org" >
 <vxbpel:VariabilityConfigurationInformation
   name="VariabilityConfigurationInformation">
  <vxbpel:ConfigurableVariationPoints>
   <vxbpel:ConfigurableVariationPoint>
    <vxbpel:Variants>
     <vxbpel:Variant name="PlanA">
      <vxbpel:RequiredConfiguration>
       <vxbpel:VPChoices name="VPChoices">
        <vxbpel:VPChoice vpname="Color" variant="White">
        </vxbpel:VPChoice>
        <vxbpel:VPChoice vpname="Transmission" variant="Manual">
        </vxbpel:VPChoice>
        <vxbpel:VPChoice vpname="Skylight" variant="Skylight_v">
        </vxbpel:VPChoice>
       </vxbpel:VPChoices>
      </vxbpel:RequiredConfiguration>
     </vxbpel:Variant>
    </vxbpel:Variants>
   </vxbpel:ConfigurableVariationPoint>
  </vxbpel:ConfigurableVariationPoints>
 </vxbpel:VariabilityConfigurationInformation>
</UserConfig>
```

6.1.4 Process instance

At run-time, the VxBPEL_ODE parses the user configuration file and derives the corresponding process instance. For the given user configuration shown in Figure 16, a process instance is derived which invokes a 'white' service under the 'colour' variation point, a 'manual' service under the 'transmission' variation point, and a 'Skylight_v' service under the 'skylight' variation point. The business process is started by a simulated client provided by FM2VxBPEL.

Up to now, we have demonstrated the application of our approach to the car assembly scenario. To further evaluate the adaptability of derived service compositions using our approach, we have randomly tested eight different user requirements. Their corresponding feature configurations and variation configurations are shown in Table 4. All these feature configurations are valid since they follow the constraints in the feature model of car assembly. For each variation configuration, the execution engine can derive a target business process instance, and then dynamically execute each instance. By analysing the log information generated during the execution, all the tested business process instances have been executed successfully.

Observations: From the above demonstration and evaluation, we observe that the various requirements as well as constraints of the car assembly scenario are well handled with the proposed feature model (answer to *question 1*). Varying requirements can be easily and efficiently met through the feature configuration and dynamic business process derivation, and the variability at different phases of compositions are trackable due to the model transformation that transfers the variability in the feature model into the variable business process (answer to *question 2*).

6.2 Performance cost evaluation

To evaluate the overhead brought by the introduction of variability management, including the customisation of feature model and the variation configuration, we collected the execution time of business processes in accordance with each configuration in Table 4. The evaluation was conducted on a computer possessing a quad core processor (3.60 GHZ), a memory of 4 GB and a Windows 7 64-bit operating system. VxBPEL_ODE was used as the process engine and each process instance was started through the simulated client of FM2VxBPEL. All user configurations were tested under the same environment. The execution time here involves the time for business process derivation and the time for the involved service's invocation. Note that differences in the execution time are mainly due to business process derivation, since the invocations of involved services are almost the same.

Observations: From Table 4, we observe that the maximum execution time is 942 ms (C_6), while the minimum is 810 ms (C_5). The performance of different business process instances is quite similar (less than one second), indicating an acceptable performance for the service composition derivation process (answer to *question 3*). In addition, the prototype provides a very useful aid for the development and execution of service compositions, and thus further enhances the practicability and efficiency of the proposed approach.

Table 4 User configurations (conf.) and corresponding execution time (T) of business processes

Conf.	*Customisation of feature model*			*Variation configuration (variation points and variants)*			*T (ms)*
	Colour	*Transmission*	*Skylight*	*Colour*	*Transmission*	*Skylight*	
C_1	Grey	Manual	N	Grey	Manual	NA	887
C_2	Grey	Manual	Y	Grey	Manual	Skylight_v	925
C_3	White	Manual	N	White	Manual	NA	902
C_4	White	Manual	Y	White	Manual	Skylight_v	913
C_5	Grey	Tiptronic	N	Grey	Tiptronic	NA	810
C_6	Grey	Tiptronic	Y	Grey	Tiptronic	Skylight_v	942
C_7	Black	Automatic	N	Black	Automatic	NA	856
C_8	Black	Automatic	Y	Black	Automatic	Skylight_v	932

6.3 Further remarks

Through the case study, we have preliminarily evaluated the effectiveness of the proposed approach. However, there are some threats to validity of our evaluation results.

- *Expressiveness of the feature model:* The demonstration of the expressive power of our approach was limited to the various simple relations in the car assembly scenario. In a complex scenario, a larger number of features and complex relations between features could be involved. The number of features has no significant impact on the expressiveness of the feature model since they can be reduced to mandatory and optional ones. Currently, our model supports five kinds of relations between features, which could be inadequate in a sophisticated situation. However, complex relations can be reduced to the combinations of those defined in our approach, although it may incur an increase in complexity of the resulting model. In this context, the feature model can be further improved if needed.
- *Representativeness of the subject scenario:* The proposed approach was evaluated through a representative scenario from the car assembly domain. Ideally, the evaluation results would be more general if more scenarios are used. However, it is hard to evaluate a software development methodology, because real-life and complex business scenarios are not always on hand. Especially, the requirements analysis and business process implementations are not publicly accessible due to commercial reason. In this context, the evaluation reported mainly serves for the demonstration purpose, which already covered the essentials of the proposed approach, including representing the variabilities of requirements and the adaptation process of variable service compositions.
- *Performance of business process:* The performance evaluation results were measured by the execution time of business process aiming to demonstrate the feasibility of the proposed approach. Obviously, the results mainly depend on the complexity of involved business processes. Furthermore, we have repeated the experiments to make them as reliable as possible.

7 Related work

In recent years, many efforts have been made to address the adaptability issue of service compositions (Razian et al., 2022). We first introduce adaptive service composition approaches based on variability management, which are closely related to our work in this paper. Then, we introduce several representative approaches based on AI techniques.

7.1 Adaptive service compositions based on variability management

This line of work focuses on handling the variability of service compositions in different ways (Rosa et al., 2017). Firstly, one group of works explore the variability of service compositions through feature modelling and configuration. Typically, this kind of methods use a feature model to represent requirement's variability, and derive service composition implementations from different feature configurations. Nguyen et al. (2016) proposed a feature-based framework to develop and provide customisable web services. In their method, a feature model is used to represent variability in user requirements, and different requirements are satisfied by mapping the feature selection to the annotation in the service interface. Similarly, Alférez et al. (2014) used a feature model to represent variable service features, and at runtime, a feature configuration is trigged and then transferred to the WS-BPEL service composition by adding or removing code fragments. Similar to these approaches, our approach also derives the implementation of service compositions from a feature model. The difference is that our approach employs an abstract service composition model to build a bridge between the feature model and implementation, which enables the traceability of variations at different phases.

Secondly, another group of works employ BPMN to model the variations of business processes. Mazzara et al. (2012) proposed a BPMN-based reconfigurable service composition approach, in which BPMN is used to model the business process with reconfigurable elements and mapping rules are then provided to derive the service composition from the BPMN model. Specifically, a reconfiguration-related activity of BPMN is mapped to a *pick* activity of WS-BPEL, and the reconfiguration regions of BPMN are mapped to scopes associated with event or termination handlers of WS-BPEL for further reconfiguration. Similarly, Terenciani et al. (2015) extended BPMN to support reconfigurations of service compositions by providing variability constructs such as variation points and variants. These approaches only focus on the adaptability of design and implementation phases, while our approach considers the full life-cycle adaptability.

Thirdly, some efforts address the adaptation issue of service compositions by providing or extending service composition languages, such as WS-BPEL. For instance, BPEL'n'Aspects (Karastoyanova and Leymann, 2009) enables the dynamic adaptation of service compositions by extending WS-BPEL, in which *advices* are web service operations, *aspects* are defined as policies, and attached to WS-BPEL processes by means of WS-policy attachments mechanism, thus the adaptation can be achieved without any modifications to the original service composition. SABPEL (Cherif et al., 2015) is another extension of WS-BPEL. It defines context policies within WS-BPEL to enable the creation of adaptable service compositions. When a context event is detected, the corresponding adaptation policies are triggered to make dynamic adjustments. In our previous work (Koning et al., 2009), we proposed VxBPEL by extending WS-BPEL with a set of variability constructs to support explicit variability implementation of

service compositions. In this study, VxBPEL has been adopted as the abstract service composition model to further enable the traceability between variable elements from requirements to implementation.

Finally, UML diagrams are used to model and manage the variability of service compositions at the architecture level. In our previous work (Sun et al., 2010), we proposed a UML profile for modelling the architectural variability of service compositions. With the modelled architectural variability, a process was further provided to manage run-time variability of the resulted VxBPEL-based service compositions. He et al. (2015) explored the architectural variability modelling of service compositions from a different perspective, and extended UML with variability elements which are then transformed to variation points and variants of VxBPEL. This study further advances the variability modelling to the requirement analysis phase and service compositions are automatically derived in a model-driven way, which significantly contributes to a more comprehensive VxBPEL-based adaptive service composition framework.

7.2 Adaptive service compositions based on AI techniques

Researchers increasingly explore popular AI techniques to address the adaptability issue of service compositions. Among them, AI planning based techniques transform service compositions to an automatic planning problem, namely finding a reachable service composition path from the initial state to the target state. For example, Bashari et al. (2018) formalised the service composition problem using a planning domain definition language (PDDL), and utilised AI planning to automatically generate a path that meet the requirements. The derived path was further converted into WS-BPEL code for execution. Rodriguez-Mier et al. (2015) used a graph-based framework for automatic service compositions, where a service relationship graph was first built based on the semantic matching of service input and output parameters, and a forward/backward graph search algorithm was then used to find the best service composition path with the least number of services.

Evolutionary algorithms-based techniques treat service compositions as a multi-objective optimisation problem and employ evolutionary algorithms to find the optimal service from available services. Specifically, various QoS properties are set as the optimisation objectives, such as response time, throughput, cost, and energy consumption. Sun et al. (2019) incorporated spatio-temporal constraints and energy consumption of services into the composition process, and used particle swarm optimisation algorithm to generate the optimal service composition. Sefati and Navimipour (2021) realised QoS-aware service compositions by integrating the ant colony optimisation algorithm with the hidden Markov model. Deng et al. (2017) first defined the energy consumption calculation model of services, and then used genetic algorithm to select services aiming to optimise the energy consumption.

Reinforcement learning based techniques model the service composition using Markov decision process (MDP) and learn the optimal service composition by maximising the cumulative reward. For instance, Ren et al. (2017) utilised Q-learning to generate the optimal service selection that satisfies QoS constraints; Yi et al. (2022) developed a deep reinforcement learning framework to achieve QoS-aware adaptability of service compositions; Liang et al. (2021) integrated the deep reinforcement learning with priority replay mechanism to achieve efficient service compositions.

In summary, the above approaches mainly focus on searching an optimal composition path or service selection solution from the perspective of optimisation, without paying attention to the concrete design and implementation of service compositions. Different from these approaches, our approach achieves the adaptability of service compositions from the perspective of variability management, considering practical development of adaptive service compositions.

8 Conclusions

We have proposed a feature-driven variability-enabled adaptive service composition approach, with the purpose of managing the full life-cycle variability of service compositions. The approach focuses on addressing the variability issue in requirements analysis, and utilises the feature model to automatically drive variability-enabled adaptive service compositions, aiming to deal with requirement changes more efficiently and systematically. This study advances the variability-based adaptive service compositions in the following aspects:

1 Feature model is adapted to the service composition domain for requirements modelling. It provides an intuitive and visualised representation for domain requirements, which enhances the comprehensibility of requirements for both the domain analyst and the process designer.

2 The MDD technique is adopted to automate the adaptive service composition process. The variability in requirements can be automatically transferred to the process implementation through model transformation, meanwhile the traceability between domain requirements and business process is persisted.

3 A prototype has been developed to visualise the modelling and configuration processes. A case study has been conducted to validated the feasibility and effectiveness of the proposed approach.

For future work, we plan to enrich the feature model to support more types of constraints between features. We also expect to conduct empirical studies to further comprehensively evaluate our approach with diverse business scenarios from more application domains, such as IT-related domains.

Acknowledgements

This work is supported by the National Natural Science Foundation of China (Grant Nos. 61872039 and 62272037), the Fundamental Research Funds for the Central Universities (Grant No. FRF-GF-19-B19), and the Open Research Fund Program of Beijing Key Laboratory on Integration and Analysis of Large-scale Stream Data (Grant No. 220190804).

References

Alférez, G.H., Pelechano, V., Mazo, R., Salinesi, C. and Diaz, D. (2014) 'Dynamic adaptation of service compositions with variability models', *Journal of Systems and Software*, Vol 91, pp.24–47.

Arcaini, P., Inverso, O. and Trubiani, C. (2020) 'Automated model-based performance analysis of software product lines under uncertainty', *Information and Software Technology*, Vol. 127, pp.1–54.

Backus, J.W., Bauer, F.L., Green, J., Katz, C., McCarthy, J., Naur, P., Perlis, A.J., Rutishauser, H., Samelson, K., Vauquois, B., Wegstein, J.H., van Wijngaarden, A. and Woodger, M. (1963) 'Revised report on the algorithmic language Algol 60', *The Computer Journal*, Vol. 5, No. 4, pp.349–367.

Bai, J., Xiao, H., Yang, X. and Zhang, G. (2009) 'Study on integration technologies of building automation systems based on Web services', *Proceedings of the 2009 International Colloquium on Computing, Communication, Control, and Management*, IEEE, Piscataway, Sanya, China, Vol. 4, pp.262–266.

Bashari, M., Bagheri, E. and Du, W. (2018) 'Automated composition and optimization of services for variability-intensive domains', *Journal of Systems and Software*, Vol. 146, pp.356–376.

Benavides, D., Segura, S. and Ruiz-Cortés, A. (2010) 'Automated analysis of feature models 20 years later: a literature review', *Information Systems*, Vol. 35, No. 6, pp.615–636.

Budinsky, F., Steinberg, D., Ellersick, R., Grose, T.J. and Merks, E. (2004) *Eclipse Modeling Framework: A Developer's Guide*, Addison-Wesley Professional, Boston.

Cherif, S., Djemaa, R.B. and Amous, I. (2015) 'SABPEL: creating self-adaptive business processes', *Proceedings of the 14th IEEE/ACIS International Conference on Computer and Information Science*, IEEE, Piscataway, Las Vegas, NV, pp.619–626.

Czarnecki, K. and Helsen, S. (2006) 'Feature-based survey of model transformation approaches', *IBM Systems Journal*, Vol. 45, No. 3, pp.621–645.

Deng, S., Wu, H., Tan, W., Xiang, Z. and Wu, Z. (2017) 'Mobile service selection for composition: an energy consumption perspective', *IEEE Transactions on Automation Science and Engineering*, Vol. 14, No. 3, pp.1478–1490.

Ezenwoye, O. and Sadjadi, S.M. (2007) 'TRAP/BPEL: a framework for dynamic adaptation of composite services', *Proceedings of the 3rd International Conference on Web Information Systems and Technologies*, Springer, Berlin, Barcelona, Spain, pp.216–221.

Gharineiat, A., Bouguettaya, A. and Ba-hutair, M.N. (2021) 'A deep reinforcement learning approach for composing moving IoT services', *IEEE Transactions on Services Computing*, DOI: 10.1109/TSC.2021.3064329.

He, X., Fu, Y., Sun, C-A., Ma, Z. and Shao, W. (2015) 'Towards model-driven variability-based flexible service compositions', *Proceedings of the 39th IEEE Annual International Computers, Software & Applications Conference*, IEEE Computer Society, Los Alamitos, CA, Taichung, Taiwan, pp.298–303.

Jamie, P.D. (2022) *How Automotive Production Lines Work* [online] https://auto.howstuffworks.com/car.htm (accessed 20 November 2021).

Karastoyanova, D. and Leymann, F. (2009) 'BPEL'n'Aspects: adapting service orchestration logic', *Proceedings of the 7th IEEE International Conference on Web Services*, IEEE Computer Society, Los Alamitos, CA, Los Angeles, CA, pp.222–229.

Koning, M., Sun, C-A., Sinnema, M. and Avgeriou, P. (2009) 'VxBPEL: supporting variability for web services in BPEL', *Information and Software Technology*, Vol. 51, No. 2, pp.258–269.

Krishnamurty, V., Natarajan, R. and Babu, C. (2013) 'Monitoring and reconfiguring the services in service oriented system using AOBPEL', *Proceedings of the 2013 International Conference on Recent Trends in Information Technology*, IEEE, Piscataway, Chennai, India, pp.423–428.

Lemos, A.L., Daniel, F. and Benatallah, B. (2015) 'Web service composition: a survey of techniques and tools', *ACM Computing Surveys*, Vol. 48, No. 3, pp.1–41.

Lian, X., Zhang, L., Jiang, J. and Goss, W. (2018) 'An approach for optimized feature selection in large-scale software product lines', *Journal of Systems and Software*, Vol. 137, pp.636–651.

Liang, H.,Wen, X., Liu, Y., Zhang, H., Zhang, L. and Wang, L. (2021) 'Logistics-involved QoS-aware service composition in cloud manufacturing with deep reinforcement learning', *Robotics and Computer-Integrated Manufacturing*, Vol. 67, p.101991.

Mazzara, M., Dragoni, N. and Zhou, M. (2012) *Implementing Workflow Reconfiguration in WS-BPEL*, Technical Report, Newcastle University, UK.

Narwane, G.K., Galindo, J.A., Krishna, S.N., Benavides, D., Millo, J-V. and Ramesh, S. (2016) 'Traceability analyses between features and assets in software product lines', *Entropy*, Vol. 18, No. 8, pp.1–31.

Nguyen, T., Colman, A. and Han, J. (2016) 'A feature-based framework for developing and provisioning customizable web services', *IEEE Transactions on Services Computing*, Vol. 9, No. 4, pp.496–510.

Peltz, C. (2003) *Web Services Orchestration: A Review of Emerging Technologies, Tools and Standards*, Technical Report, Hewlett-Packard Company.

Razian, M., Fathian, M., Bahsoon, R., Toosi, A.N. and Buyya, R. (2022) 'Service composition in dynamic environments: a systematic review and future directions', *Journal of Systems and Software*, Vol. 188, p.111290.

Ren, L., Wang, W. and Xu, H. (2017) 'A reinforcement learning method for constraintsatisfied services composition', *IEEE Transactions on Services Computing*, Vol. 13, No. 5, pp.786–800.

Rodriguez-Mier, P., Pedrinaci, C., Lama, M. and Mucientes, M. (2015) 'An integrated semantic web service discovery and composition framework', *IEEE Transactions on Services Computing*, Vol. 9, No. 4, pp.537–550.

Rosa, M.L., Aalst, W.M.V.D., Dumas, M. and Milani, F.P. (2017) 'Business process variability modeling: a survey', *ACM Computing Surveys*, Vol. 50, No. 1, pp.1–45.

Sebastián, G., Gallud, J.A. and Tesoriero, R. (2020) 'Code generation using model driven architecture: a systematic mapping study', *Journal of Computer Languages*, Vol. 56, pp.1–11.

Sefati, S. and Navimipour, N.J. (2021) 'A QoS-aware service composition mechanism in the internet of things using a hidden-markov-model-based optimization algorithm', *IEEE Internet of Things Journal*, Vol. 8, No. 20, pp.15620–15627.

Sun, C-A., Rossing, R., Sinnema, M., Bulanov, P. and Aiello, M. (2010) 'Modeling and managing variability of web service-based systems', *Journal of Systems and Software*, Vol. 83, No. 3, pp.502–516.

Sun, C-A., Wang, P., Zhang, X. and Aiello, M. (2014) 'VxBPEL_ODE: a variability enhanced service composition engine', *Proceedings of APWeb 2014 Workshops, LNCS 8710*, Springer, Cham, Changsha, China, pp.69–81.

Sun, C-A., Wang, Z., Wang, K., Xue, T. and Aiello, M. (2019a) 'Adaptive BPEL service compositions via variability management: a methodology and supporting platform', *International Journal of Web Serivices Research*, Vol. 16, No. 1, pp.37–69.

Sun, M., Zhou, Z., Wang, J., Du, C. and Gaaloul, W. (2019b) 'Energy-efficient IoT service composition for concurrent timed applications', *Future Generation Computer Systems*, Vol. 100, pp.1017–1030.

Sun, C-A., Xue, T. and Hu, C. (2013) 'VxBPELEngine: a change-driven adaptive service composition engine', *Chinese Journal of Computers*, Vol. 36, No. 12, pp.2441–2454.

Sun, C-A., Zhang, Z. and Zhang, X. (2018) 'A variation model-based reusable and customizable SaaS development approach', *Chinese Journal of Software*, Vol. 29, pp.3435–3454.

Terenciani, M., Paiva, D.M.B., Landre, G. and Cagnin, M.I. (2015) 'BPMN* – a notation for representation of variability in business process towards supporting business process line modeling', *Proceedings of the 27th International Conference on Software Engineering and Knowledge Engineering*, Wyndham Pittsburgh University Center, USA, Pittsburgh, USA, pp.227–230.

Urbieta, A., González-Beltrán, A., Mokhtar, S.B., Hossain, M.A. and Capra, L. (2017) 'Adaptive and context-aware service composition for IoT-based smart cities', *Future Generation Computer Systems*, Vol. 76, pp.262–274.

Valderas, P., Torres, V. and Serral, E. (2022) 'Modelling and executing IoT-enhanced business processes through BPMN and microservices', *Journal of Systems and Software*, Vol. 184, p.111139.

Wang, H., Hu, X., Yu, Q., Gu, M., Zhao, W., Yan, J. and Hong, T. (2020) 'Integrating reinforcement learning and skyline computing for adaptive service composition', *Information Sciences*, Vol. 519, pp.141–160.

Xiao, Z., Cao, D., You, C. and Mei, H. (2011) 'Towards a constraint-based framework for dynamic business process adaptation', *Proceedings of the 8th IEEE International Conference on Services Computing*, IEEE Computer Society, Los Alamitos, CA, Washington, DC, USA, pp.685–692.

Yi, K., Yang, J., Wang, S., Zhang, Z. and Ren, X. (2022) 'PPDRL: a pretraining-and-policy-based deep reinforcement learning approach for QoS-aware service composition', *Security and Communication Networks*, p.8264423.

Yu, W., Zhao, H., Zhang, W. and Jin, Z. (2016) 'A survey of the feature model based approaches to automated product derivation', *Chinese Journal of Software*, Vol. 27, No. 1, pp.26–44.