The Journal of Systems and Software 83 (2010) 1123–1136

Contents lists available at ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

A feature-oriented approach for developing reusable product line assets of service-based systems

Jaejoon Lee [a,*], Dirk Muthig [b,1], Matthias Naab [c,2]

[a] School of Computing and Communications, Lancaster University, Lancaster, United Kingdom
[b] Lufthansa Systems, Germany
[c] Fraunhofer Institute for Experimental Software Engineering (IESE), 67663 Kaiserslautern, Germany

ARTICLE INFO

Article history:
Received 10 February 2009
Received in revised form 1 September 2009
Accepted 27 January 2010
Available online 4 February 2010

Keywords:
Software product line engineering
Feature-oriented
Service-based systems
Software architecture
Software architecture styles

ABSTRACT

Service orientation (SO) is a relevant promising candidate for accommodating rapidly changing user needs and expectations. One of the goals of adopting SO is the improvement of reusability, however, the development of service-based system in practice has uncovered several challenging issues, such as how to identify reusable services, how to determine configurations of services that are relevant to users' current product configuration and context, and how to maintain service validity after configuration changes. In this paper, we propose a method that addresses these issues by adapting a feature-oriented product line engineering approach. The method is notable in that it guides developers to identify reusable services at the right level of granularity and to map users' context to relevant service configuration, and it also provides a means to check the validity of services at runtime in terms of invariants and pre/post-conditions of services. Moreover, we propose a heterogeneous style based architecture model for developing such systems.

1. Introduction

Service orientation (SO) is a relatively new paradigm for software development: systems are no longer developed, integrated, and released in a centrally synchronized way, but services are developed and deployed independently and separately in a networked environment, as well as composed as late as at runtime (Erl, 2008; Zhu, 2005; Dan et al., 2008).

Over the last few years, the emergence of many application domains that could benefit from the idea of service orientation had been observed. This is mainly due to the rapid growth of the Internet in terms of size and speed, and ambient hardware devices that are small but have enough computing power to provide services. For example, diverse pieces of end user equipments are capable of interacting on their own in a "virtual office" domain – either because certain persons are identified, messages are received from other equipments, or based on status of higher-level business workflows. Additionally, office equipment will provide services to its potentially mobile users that are useful in their current context (i.e., their role and responsibilities, the active workflows, and the available technical infrastructure).

While these application domains benefit from SO, there had not been a great attention about the *reusability* of software artifacts. As the domain matures, however, the need for a systematic approach for developing reusable assets of such service-based systems is being recognized in the literature (Zhu, 2005; Dan et al., 2008; Lee et al., 2008).

The resulting method thus needs to tackle several challenging issues organizations practically experience while developing reusable assets of service-based systems. These issues include:

- the identification of reusable services: SO facilitates an easy composition (orchestration) of services. This capability is one of the key benefits of service-based system, however, it also results in populating multiple similar services, which is the main obstacle to establishing a systematic reuse;
- the determination of service configurations that are relevant to users' current product configuration and context: different from static variation control of traditional software product line approaches, we should be able to control service configurations depending on each user's current needs and available resources at runtime; and
- the provision of a means to maintain service validity during workflow transactions: the capability of autonomous configuration change requires a mechanism by which we can check whether current services are valid for a particular configuration and user's context.

* Corresponding author. Tel.: +44 1524 510359; fax: +44 1524 510492.
E-mail addresses: j.lee@comp.lancs.ac.uk (J. Lee), dirk.muthig@lhsystems.com (D. Muthig), matthias.naab@iese.fraunhofer.de (M. Naab).
[1] Tel.: +49 69 696 32868; fax: +49 69 696 9832868.
[2] Tel.: +49 631 6800 2249; fax: +49 631 6800 9 2249.

0164-1212/$ - see front matter © 2010 Elsevier Inc. All rights reserved.
doi:10.1016/j.jss.2010.01.048

Note that one important assumption in this paper is that the developed assets are under the control of a developing organization. That is, we focus on how to develop reusable assets of service-based systems. Therefore, a service provider is also developed and deployed by the organization with the reuse of product line assets and we do not imply a third-party service provider.

In this paper, we extend (Lee et al., 2008) by (1) focusing on *improving reusability* of service-based systems, (2) adding *context* definition and specification, (3) explaining *product engineering process*, and (4) refining our case study to *provide more technical details* to readers. We adapted a feature-oriented product line engineering approach, which has been applied successfully for establishing software reuse in practice (Kang et al., 2002; Lee and Kang, 2006). Our method is the novel fusion of the two research themes of services and software product line engineering: achieving flexibility of network based systems through service orientation, but still managing product variations through product line engineering techniques.

It is based on the feature analysis technique (Lee and Muthig, 2006; Lee et al., 2002) that enables us to identify reusable services of a service-based system. The method is notable in that it guides developers to identify reusable services of a service-based system and to map users' context to relevant service configuration, and it also provides a means to check the validity of services in terms of invariants and pre/post-conditions of services. Moreover, we propose a heterogeneous style based architecture model for the systematic development and variation control of such systems.

1.1. Related work

Achieving reusability is an often stated goal of SOA in the literature. In spite of this, only a few approaches and ideas exist in research publications. While the approach in this paper concentrates on achieving reusability by means of proper identification and specification of services using product line technologies, Zhu (2005) follows another approach. There, reusability is claimed to be achieved by the structure of systems and the interaction mechanisms. This mainly means the availability of a service repository and the concepts for discovering, negotiating, and binding services. Dan et al. (2008) address reuse of services very explicitly. The main idea is to apply different practices of SOA governance to address aspects like terminology, service discovery and creation, and service entitlement. Mainly, the paper details and discusses and details the challenges of reuse in SOA. Our approach can be seen as a constructive answer to the challenges in the area of service creation.

IBM developed a method for the development of SO systems called 'Service-Oriented Modeling and Architecture' (Arsanjani, 2004; Arsanjani and Allam, 2006). It provides guidelines for three steps towards SO systems: Identification, specification, and realization of services, flows, and components. Most details in Arsanjani (2004) are related to the identification of services. There, a combination of three complementary ideas is proposed: First, the domain of the respective software systems is analyzed and decomposed. Second, existing legacy systems are explored in order to discover parts to be reused as services. Third, business goals are taken into account to complete the identification of services.

The first and third ideas are also reflected in our approach. Our approach supports the service identification by the feature orientated analysis and thus we could also analyze various relationships (i.e., aggregation, generalization/specialization, and binding) among identified services. The approach of IBM further suggests organizing services in a hierarchy of services of different granularity. Our approach adds the dedicated layer of molecular services that form reusable assets in the specific domain. According to the respective domain, the molecules would be composed in different ways to optimally fit the requirement of reuse. Thus, reuse becomes easier by only selecting from a rather small number of assets with well-tailored granularity. The concept of flows of services is mentioned to be important in Arsanjani (2004), however, there are no details about the identification or specification of these flows. Our approach incorporates the defined molecular services as the building blocks of which to orchestrate workflows.

To a certain extent SOA and SPLE (Software Product Line Engineering) share common goals like reusability in order to achieve economic benefits. The relationship among the two and the potential for mutual benefits has been recently explored. So far, this exploration is mostly done at the general level (e.g. Helferich et al., 2007) in order to identify common ideas and differences. Especially feature-orientation, which is often used as an approach to capture commonalities and variabilities in SPLE has been analyzed for the support of service orientation. Challenges identified in this area are presented in Apel et al. (2008). Our paper presents a very concrete way how feature-orientation is applied in order to improve the reusability of services.

Another approach of using feature-oriented analysis to identify services for a SO system is described in Chen et al. (2005). Their main focus is reengineering towards SO systems. Therefore, they claim to do a feature analysis of the particular system and use the result as input for the service identification. What is missing there is concrete guidelines how to come up with services of the right granularity. It is only stated that services should be as coarse-grained as possible. The lack of elements putting more structure on the feature model, like feature binding units, makes service identification more complex.

In the literature, there are a number of languages to express service orchestrations. In the field of Web Services, the most popular technology for realizing SO systems, BPEL4WS (Business Process Execution Language for Web Services) (Juric et al., 2003) is well known. It represents a language to specify orchestrations of services that are then accessible as higher-level services.

In our approach, the orchestrated services are described as workflows. A further concept we transferred to service composition is 'Design by Contract' (Meyer, 1991). This means to enrich the composition language and service description by pre/post-conditions and invariants that can be automatically verified. Hence, the reliability of service composition, static as well as dynamic, can be improved by checking the correct usage of services. Thus, the reusability of services with advanced description is improved since automatic checks can reduce the number of feasible candidate services, which makes selection easier.

For the development of service-oriented systems, a number of reference architectures have been proposed (Georgantas et al., 2005; Arsanjani et al., 2007a,b; OASIS Reference Architecture; Durvasula, 2007). Typically, the focus in these reference architectures is on the description of the overall system and especially on the organization and orchestration of services on the provider's side. That is, only less documentation is available how applications are to be designed that are settled in a service-oriented architecture (Krafzig et al., 2005). Such applications are mostly characterized as service consumers, but their internals are not subject of the reference architecture. In contrast, our architectural model emphasizes the service consumers and combines architectural styles to achieve the domain-specific design goals.

1.2. Approach overview

Product line processes generally aim at a systematic exploitation of common characteristics and predicted variations among products of the same family (Arsanjani, 2004; Clements and

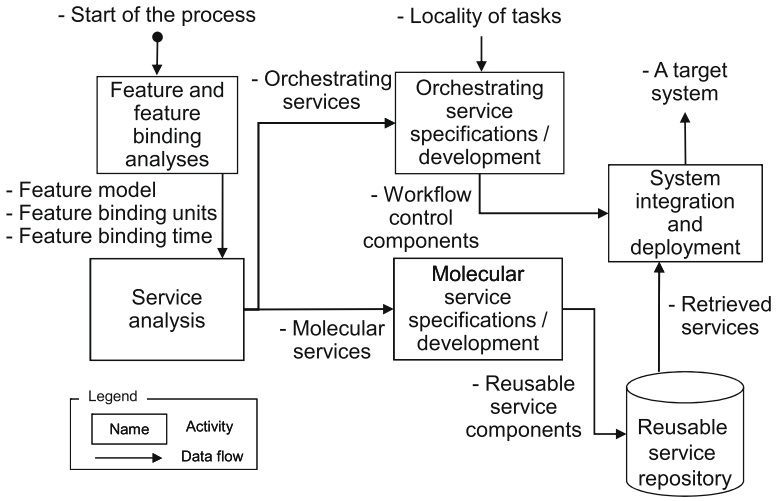

Fig. 1. Activities of the approach.

Northrop, 2002; Pohl et al., 2005). The key idea is thereby to split the overall lifecycle into two main phases: application and family engineering. Each instance of application engineering constructs and maintains one particular product. Family engineering constructs and evolves the reuse infrastructure that is supposed to make application engineering more efficient. The input to family engineering is the specification of a product family (i.e., the product line scope), whose members are produced by application engineering projects.

The method has thus to guide through the construction of a reuse infrastructure, on the one hand, that consists of generic services optimized for the particular family of envisioned products. On the other hand, it also must guide construction and evolution of family members, which are heavily based on existing services (i.e., heavily reusing existing services).

Fig. 1 shows activities and their relationships of the technical component presented. These activities are executed iteratively; the arrows in Fig. 1 indicate the flow of data and which work products are used by each activity.

A feature analysis organizes product family features into an initial model, which is then refined by adding design features such as operating environments, domain technologies, or implementation techniques. Within the feature model, the subsequent binding analysis identifies binding units and determines their relative binding times among each others (Lee and Kang, 2004).

The service analysis consumes the results of these analyses. Each binding unit is further analyzed to determine its service category (i.e., orchestrating service or molecular service) with respect to the particular family at hand. We assume here families whose variations can be described best by variations in workflows executed by the system users. Additionally, the context and the technical infrastructures available vary and thus dynamic reconfigurations of product variants are expected.

In our approach, the mass of low level services are grouped into richer services as required by the family. We call these services as 'molecular'[3] services. Note that each product family has thus its own specific set of molecules, the reusable services for constructing family members. Due to the definition of those molecules based on product line processes, molecular services are more reusable than low level services in the context of a particular product family. On the other hand, the high level services, that we call orchestrating services, are specified first as workflows and their constituting tasks. Then, their pre/post-conditions, invariants, and service interfaces are specified. Finally, the system integration and deployment activity form a product and the orchestrating services provide services to users by integrating and parameterizing the molecular services at runtime.

1.3. Outline

The remainder of this paper is organized as follows: Section 2 generally introduces feature analysis as a key activity within product line engineering to identify commonalities and predicted variabilities of a product line. Section 3 then describes in detail the one major step of our method, namely the analysis of services, their identification and orchestration. Section 4 presents then the other major part of our method that is the underlying architectural style that defines the structures of service-based product lines as proposed by our approach. Section 5 explains how the artifacts of each process are put together to deliver a product to a customer by reusing previously developed services. Section 6 demonstrates the approach by presenting a case study from the office domain, which has been created under a research theme we call the virtual office of the future. Section 7 concludes the paper by discussing what has been achieved so far, as well as outlining the planned future work.

2. Feature analysis

In this section, activities of feature analysis, which includes feature modeling and feature binding analysis are introduced. For illustrating the approach presented in this paper, we selected a case study in the domain of the virtual office of the future (VOF). The VOF product family consists of systems, which control and manage collections of devices to provide any-time any-where office environments (VOF). In this paper, we limit ourselves to the VOF features in Table 1.

Feature modeling is the activity of identifying externally visible characteristics of products in a product line and organizing them into a model called a feature model (Lee et al., 2002). (Fig. 2 is a feature model of the VOF product line.) The primary goal of feature modeling is to identify commonalities and differences of products in a product line and represent them in an exploitable form, i.e., a feature model.

[3] In chemistry, a molecule is defined as 'a sufficiently stable electrically neutral group of at least two atoms in a definite arrangement held together by strong chemical bonds' (Union of Pure and Chemistry, 1994) We adopt this notion of molecular, as a molecular service represents a unique service, which will be used as-is without a further decomposition in a particular domain.

Table 1
Product features of VoF product line.

Product feature	Description
Follow Me (FM)	In the VOF product line, information on users' (physical) locations is important to provide context-relevant services. This feature detects physical location of a user by using various locating devices such as access points (AP) of wireless LAN, personal ID cards and RFID. A user's location is updated when events from the user are detected or at pre-determined intervals. One of the FM's optional features is Automatic Log-on, which allows a user to access facilities (e.g., computers, printers, rooms, etc.) of an office building without manual operations for authentification. This feature must be bound at runtime only if (1) FM is selected for the current product configuration, (2) the requesting user's job function is a manager or a director, and (3) a RFID-based locating device is available nearby
Virtual Printer (VP)	This feature selects the nearest printer to a user with a most appropriate printing quality at the moment when the service is requested
Smart Business Trip (SBT)	The smart business trip feature supports planning, approving, preparing, and reporting a business trip. After a traveling employee triggers this service, relevant tasks for various stakeholders (e.g., a manager who has the authority to approve the trip and a secretary who makes reservations for hotels and transportations) are invoked automatically. The system should recognize the context of each stakeholder and configure/bind services to be best fit into current situations. Suppose, for example, that a user needs to print a file, then an appropriate printer is selected automatically and its location is notified to the user by using the *Virtual Printer* feature
Smart Fax (SF)	A smart fax feature supports on-line fax send/receive services. A user can send a fax on-line by providing a destination fax number, recipient's name and organization, and preferred transmission time, after preparing a fax document at her/his computer. When a fax is received, this feature can also detect a recipient of the fax by using an OCR function and send a notification to its recipient via email or SMS

Common features among different products in a product line are modeled as mandatory features (e.g., *Virtual Printer* and *Smart Fax*), while different features among them may be optional (e.g., *Automatic Log-on*) or alternative (e.g., *AP-based* or *RFID-based User Positioning Method*). Optional features represent selectable features for products of a given product line, and alternative features indicate that no more than one feature can be selected for a product. In a feature diagram, these features are organized by using three types of relationships: *composed-of, generalization/specialization, and implemented-by*. For example, *Smart Fax* is composed of *On-line Fax Send*, *Recipient Recognition*, and *Recipient Notification*. Composition rules supplement the feature model with mutual dependency and mutual exclusion relationships which are used to constrain the selection from optional or alternative features. In the VOF product line, *Recipient Notification* requires *Recipient Recognition* and they should be selected together.

Once we have a feature model, it is further analyzed through feature binding analysis. Feature binding analysis consists of two activities: feature binding unit identification and feature binding time determination. Feature binding unit identification starts with identification of service features. A service feature represents a major functionality of a system and may be added or removed as a service unit. In VOF, *Smart Fax* and *Smart Business Trip* features are examples of service features.

Because a feature binding unit contains a set of features that need to be bound together into a product to provide a service correctly and share a same binding time, a product can be considered as a composition of feature binding units. For instance, *Smart Fax* and *On-Line Fax Send* should be incorporated into a product at the same binding time (e.g., compile time, installation time, or runtime) to provide the *Smart Fax* service properly.

Through the binding analysis, we could have the following information on a service-based product line:

- candidate reusable services: a set of feature within a binding unit should be together to provide a service and, therefore, this feature group could be a right granularity for developing reusable services;
- dependency among services: the link between binding units represents binding time dependency (i.e., a parent binding unit must be bound beforehand for the binding of its descendent binding units) and this should be consistent with their implementation; and
- variation points for runtime binding: through the binding time analysis, we can make an early decision on which service should be deployed and bound into a product at runtime. Such services and their constituting features should be designed to support the binding at runtime.

In the next section, it is explained how the identified candidate services (i.e., feature binding units) are further classified and refined.

3. Service analysis

Through the previous activities, we now have a feature model and feature binding information, which provides an insight into a targeting domain in terms of product features, basic units of binding, and their binding time. Then, the feature model is refined and restructured by introducing two distinctive service characteristics: behavioral (workflow) and computational (tasks) service characteristics.

The primary objective of introducing these two different criteria of the feature classification is to address the first difficulty that we discussed: avoiding rapid population of similar services but capturing reusable services of a service-based system. Based on our observations on service-based systems, we identified two different characteristics of services: a behavior oriented service, which is very likely modified for each organization or user's needs (e.g., workflow for a business trip process), and a computation oriented service, which can be reused across multiple product configurations (e.g., a user localization service).

A behavior oriented service is mainly to define a certain sequence of tasks, i.e., workflows. We call services in this category as orchestrating services, as their main role is the composition of other services in a harmonious way. A computation oriented service is to provide computational outputs (i.e., a pre-defined task to be conducted by an IT system or a person) in response to given inputs. We call services in this category as molecular services, as they are the basic building blocks and will be reused as-is by orchestrating services. Details of services that belong to each category are explained in the following sections. (See Fig. 3 for the refined feature model with the two service layers.)

The following sections describe the details on orchestrating and molecular services with examples.

3.1. Orchestrating service

An orchestrating service has the responsibility to reflect a workflow, which a user of a system can conduct or trigger, and the coordination of system functionalities. In the VOF product line, an orchestrating service usually consists of a whole sequence of

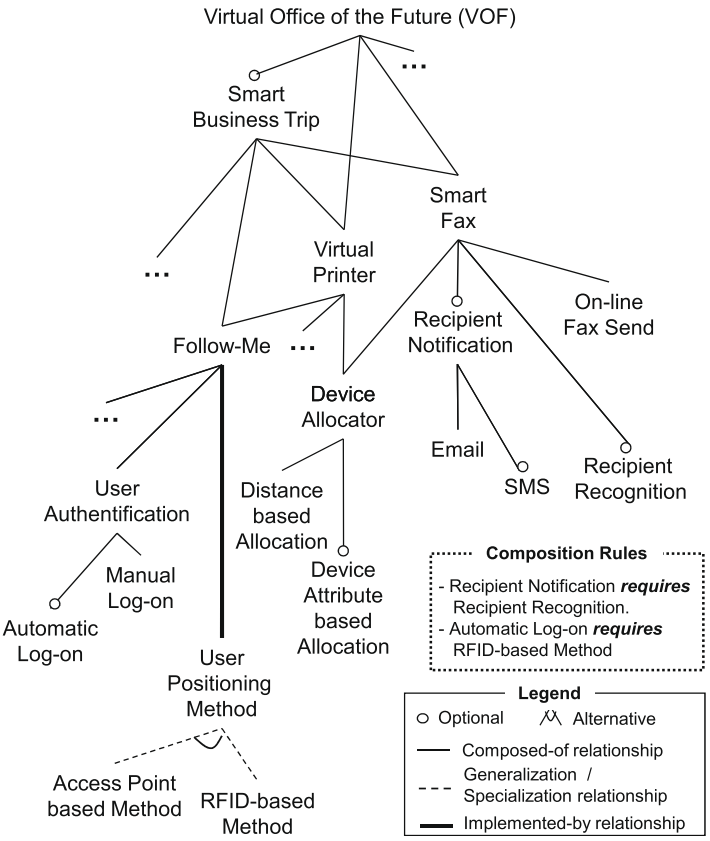

Fig. 2. Feature model of VOF.

interactions between one or more users and the coordination among related molecular services.

3.1.1. Basics for the definition of orchestrating services

The basic constructs needed for defining orchestrating services are the following: First, tasks are the basic building blocks of a workflow. They can represent the interaction of the system with a user, e.g. to collect information in a workflow, or they can represent triggering some computational function in form of another service. Second, decisions are an important construct to allow for the definition of a more complex control flow. The decisions can refer to all data available in the workflow or particularly defined context data. Third, parallelizing of the control threads is possible to allow for the parallel execution of tasks. Forth, the concepts of users and roles are important. A workflow can involve and coordinate the work of multiple users. To be independent of individual users, roles are defined. Then tasks can be assigned to appropriate and available people, depending on the role definitions.

3.1.2. Extended support for reliability

For orchestrating services, correctness of their overall control behavior is the foremost concern. For example, providing an expensive color-printing service with proper authorization is critical in the virtual office environment. Therefore, we extended the BPEL (Juric et al., 2003) workflow definition language with pre/post-conditions and invariants to enhance the reliability of specifications.

Fig. 4 shows a workflow specification example for the *Smart Business Trip* service. Each orchestrating service has pre/post-conditions and invariants. In this example, a user should be logged in to trigger the service and the workflow is completed only after the user submits a postmortem report about her/his business trip. Also, the invariants (i.e., the user is employed and the business trip is not cancelled) should hold through the whole workflow process. (See the text box at the mid-left portion of Fig. 4.) Whenever the invariants become invalid, the workflow is terminated with proper notifications to relevant stakeholders or other behavior for the resolution of the problem situation can be defined.

Moreover, each task of the workflow can be specified with its pre/post-conditions and invariants. That is, processing a single task can involve several interactions between a user and the system and thus it is valid to define invariants for tasks. For example, a secretary should achieve the access right to organizational data such as the charged project's budget information and the traveler's bank account number to proceed with the 'reservations' task. These conditions can be defined as the precondition of the reservation task and checked when a secretary is assigned for the task. Also the consistency of invariants between a workflow and its constituting tasks should be checked when an orchestrating service is specified.

3.1.3. Extended support for mobility

Another important concern in today's information systems is the increasing mobility of users: users want to work with a system at any time at any place. Although the coverage of wireless technology continuously increases, there are still many situations that require working in disconnected mode. For working in a disconnected environment, the availability of services and the appropriate data on the user's device should be identified. This is the second important aspect that we address at the level of orchestrating services. Therefore, the term *locality* is used. A task is called local if all information needed by the responsible person is locally available on this person's computing device. A sequence of tasks is called local if only one person is responsible for these tasks (i.e. no switch between roles) and if all tasks in this sequence are local.

The VOF product line particularly relies on this ability of mobile and disconnected working. For example, a person on a business

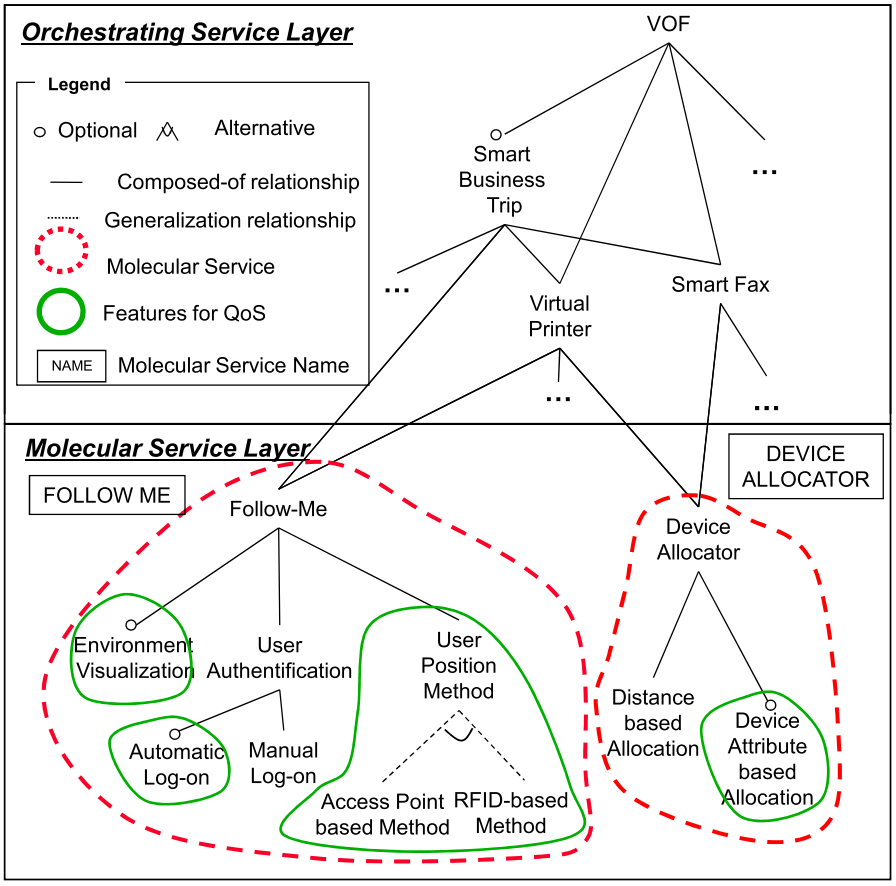

Fig. 3. A refined feature model based on two service categories.

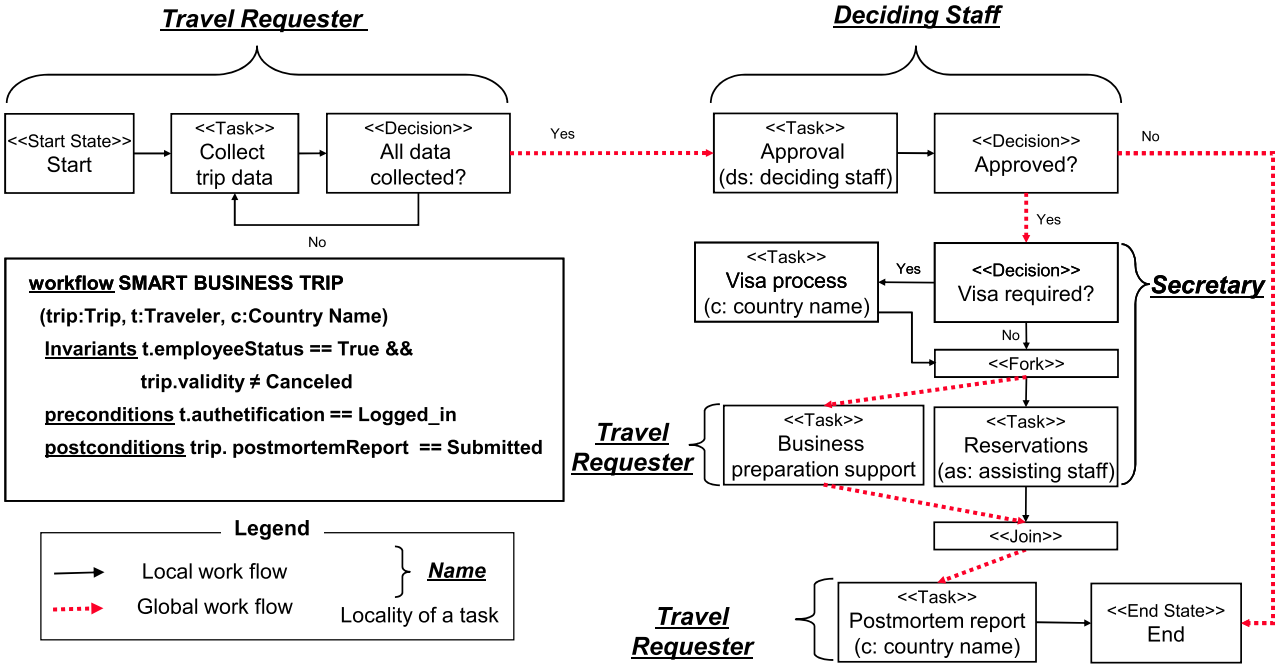

Fig. 4. An example of workflow specification for an orchestrating service: smart business trip.

trip can process all its tasks belonging to a workflow as long as they are local. That is, no global coordination with the overall system or other users is necessary. After a reconnect to the other system parts, the global coordination can go on and the results produced locally can be shared with people to trigger them for further processing.

```
1: molecular service FOLLOW ME (user User)
2: invariants user.employeeStatus == true
3: precondition user.authentification == logged_in
4: postcondition none;
5:   option Environment Visualization
6:     binding_time runtime
7:     precondition user.device == desktop ∨ notebook
8:     postcondition none;
9:   option Automatic Log-on
10:    binding_time runtime
11:    precondition user.rank == director ∨ manager and
12:        RFID bases user location method == available
13:    postcondition user.access == granted ∨ rejected;
```

Fig. 5. An example of molecular service specification.

Next, the identification and specification of molecular services are explained.

3.2. Molecular services

The identification of molecular services with right granularity is the key factor to enhance reusability of the service-based system development. Molecular services are the basic units for reuse and orchestrating services should be able to compose them as-is through their interfaces during development time or runtime.

3.2.1. Basic properties of molecular service

For their identification, feature binding units are analyzed and refined with consideration of the following guidelines. A molecular service should be:

- self-contained (local control and local computation);
- stateless from service user's point of view;
- provided with pre/post-conditions; and
- representative of a domain-specific service.

The first three guidelines are to decouple service consumers from providers. Based on these guidelines, a service consumer only needs to know the service providers' interfaces and their conditions for use. This means that any changes (performance improvements, bug patches, etc.) within an identified molecular service should be encapsulated. The last guideline is the key factor to determine the right granularity of a molecular service based on the feature binding unit and time information, and domain experts' professional judgment.

In our case study, for instance, the feature binding units related to *Follow Me* and its descendent feature binding units are identified and reorganized as the *FOLLOW ME* molecular service in Fig. 3. The rationale for this determination is as follows:

- the *Follow Me* feature is a mandatory service for every user of the VOF product line,
- each localizing device (e.g., RFID, access points of wireless networks, etc.) uses different localization techniques, but their expected outputs are the same (e.g., a user's physical location),
- the implementing algorithms for localization evolve rapidly to improve their accuracy, and
- it is a computation oriented service without any workflows in it.

Based on this decision, the *FOLLOW ME* molecular service is designed and implemented to provide the user localization service to the orchestrating services, if they abide by the pre/post-conditions of *FOLLOW ME*.

A molecular service may have QoS parameters, which are identified during the feature binding analysis in terms of optional or alternative features. That is, by using different functionality with different properties, the overall services can exhibit different QoS levels. For example, the *User Position Method* feature has two alternatives (e.g., *Access Point based Method* and *RFID-based Method*) and their levels of accuracy are different (e.g., the error range of the RFID-based method is less than 1 meter, whereas the error range of AP-based method is less than 10 m). Depending on available devices near a user, one of the alternative positioning methods is selected and used.

3.2.2. Specification of molecular services

In our approach, each molecular service is specified by using a text-based specification template and Fig. 5 shows the specification of *FOLLOW ME*. (The characters in the bold font are reserved words for the specification.) The *FOLLOW ME* service is for the current employees, who passed the authentification and logged in. Also, the *Automatic Log-on*, which is optional for higher quality of the service, is only available at runtime when the requesting user's job function is director or managers, and a RFID device is available near by. (See the lines 9–13 for the specification of optional feature *Automatic Log-on*.)

In this section, concepts and guidelines for analyzing and specifying molecular services are explained. The next section introduces context analysis and specification for the control of runtime variation.

3.3. Context analysis and specification

The context awareness is one of the important capabilities for dynamic service/product reconfiguration (Hallsteinsen et al., 2006; Bencomo et al., 2008). The context analysis of this paper is adopted from Lee and Muthig (2008), which starts with identifying *contextual parameters* of a product line. A contextual parameter is defined as an environmental element that has a piece of

Table 2
Contextual parameter definition

Contextual parameters	Attributes		
	Type	Sampling rate	Validity (a valid range of value or a set of valid values)
Privilege Level (P)	String	Log-in time	P = "Director" ∨ "D-Head" ∨ "Manager" ∨ "Scientist" ∨ "Visitor" ∨ "Administrator"
Available User Localizer (AL)	String	60 s	AL = "RFID" ∨ "AP"
Device (D)	String	Log-in time	D = "Desktop" ∨ "notebook" ∨ "PDA" ∨ "PHONE"

information about a system's context (e.g., current location of a user, battery remaining time, etc.). Once contextual parameters are identified, we refine them by defining attributes of each parameter. The attributes may include data type, sampling rates, and validity conditions. (See Table 2 for a part of contextual parameter definitions for *VOF*.) In the *Type* column, the types of contextual parameter values are defined. The *Sampling Rate* defines how often the contextual parameters should be checked. A contextual parameter may be valid only if its value is within a pre-defined range or a set of values: such conditions are defined in the *Validity* column. The validity conditions of each contextual parameter should be satisfied before a contextual parameter is used to detect contextual changes.

Then, each situation is specified as a logical expression of contextual parameters. When a situation is recognized, it triggers a dynamic reconfiguration. For instance, a *Automatic Log-on Allowed* situation is true when *Privilege Level* (*P*) is *Director* and the *Available User Localizer* (*AL*) is *RFID*. For this situation, the *Automatic Log-on* can be bound and activated. This means that a higher quality of *FOLLOW ME* molecular service (i.e., *FOLLOW ME* with *Automatic Log-on* enabled) is provided, when the user is *Director* and equipped with *RFID* device.

The next section introduces an architecture model for the development of these services.

4. A Heterogeneous style based architecture model

A software architectural style (or a style in short) captures a recurring form and its variations of software system design (Clements et al., 2002). A style comes along with a set of constraints that it satisfies and this information is essential for software designers to make a right architectural decision. Also, style specific tools and implementations may be available for developing a system that uses the style. It should be also noticed that a system seldom comprises a single style, but rather a set of styles for different parts of the system design. Therefore, it is important to apply styles based on explicit rationale (i.e., design goals) and maintain consistency among them.

For the development of the VOF system, we propose a HEterogeneous style based ARchiTecture (HEART) model, which consists of three decomposition levels. In the following sections, we explain the design goals, meta-models of architectural styles used at each decomposition level to achieve the goals, and an instance of the meta-model for the VOF system.

4.1. Design goals

While we explored various design issues for developing systems for future office environments, we identified the quality attributes flexibility and scalability as very important. The following main design goals concretize the quality attributes in our context and serve as input for the construction of the architectural model:

1. Support for late binding of networked services to their consumers: this is to achieve the runtime flexibility, which is one of the main ideas of SO. By networked services we mean any entities that can be developed and deployed independently but their binding to their consumers occurs at runtime when the consumers request. In the VOF product line, shared business peripherals (e.g., printers, fax machines, etc.), a workflow engine that processes the global transaction are the examples. Also, the system scope should be able to scale up through the Internet.
2. Support for mobile products: In the future office environments, people may use mobile devices and these devices may not have a continuous connection to central infrastructures. Nevertheless, the devices should provide as much functionality as possible.
3. Support for four main functionalities of a service consumer: a service consumer should be able to maintain connectivity to the system domain, recognize current context, interact with a user, and manage multiple active services at a certain moment. Moreover, the priorities among services may vary depending on users' current situations. This implies that a developer should be able to define multiple concurrent processes as well as their priorities.
4. Support for our notion of service classification: we analyzed two distinct service characteristics (i.e., orchestrating and molecular services) to identify services with right granularity and clarify their interactions. The architecture design should facilitate these concerns for identifying and deploying architectural components.
5. Support for dynamic reconfiguration: a product should be able to reconfigure and parameterize its services depending on recognized situations and available resources at the moment.
6. Support for product line variation control: we should be able to control the variation of each product so that each user can have her/his own product configuration.
7. In the following section, we explain our proposed architecture model and how these design goals are achieved.

4.2. The HEART model

The HEART model consists of three decomposition levels and each level addresses specific design goals listed above by adopting architecture styles. The top level supports the first and second design goals by adopting the service-oriented style (Michlmayr et al., 2007). (The upper portion of Fig. 6 shows the meta-model of the style.)

Service providers and consumers can join and leave the system scope (i.e., domain) independently and the information broker takes care of their authentification, registration, retrieval. Also, the trust relationship can be established between information brokers so that the service consumers can join the trusted domain and access services in that domain. Note that we did not impose any constrains on service providers, as long as they abide by the interfaces with the information broker and service consumers.

The lower portion of Fig. 6 shows an instance of the service-oriented style. In this example, three printing service providers are deployed: *Guest Printer*, *Color Printer*, and *Default Printer*. Each service provider has its unique profile, such as supported paper sizes and color-printing capability, and it registers this information to the *A Domain* information broker when it is ready to provide the printing service. Also, three service consumers are deployed: *Director*, *Scientist*, and *Guest*. Each name represents its role and its accessibility to the service providers are decided by the role. For example, *Director* can access all printer services, while *Guest* can only access the *Guest Printer* service provider. This information is maintained by the information broker.

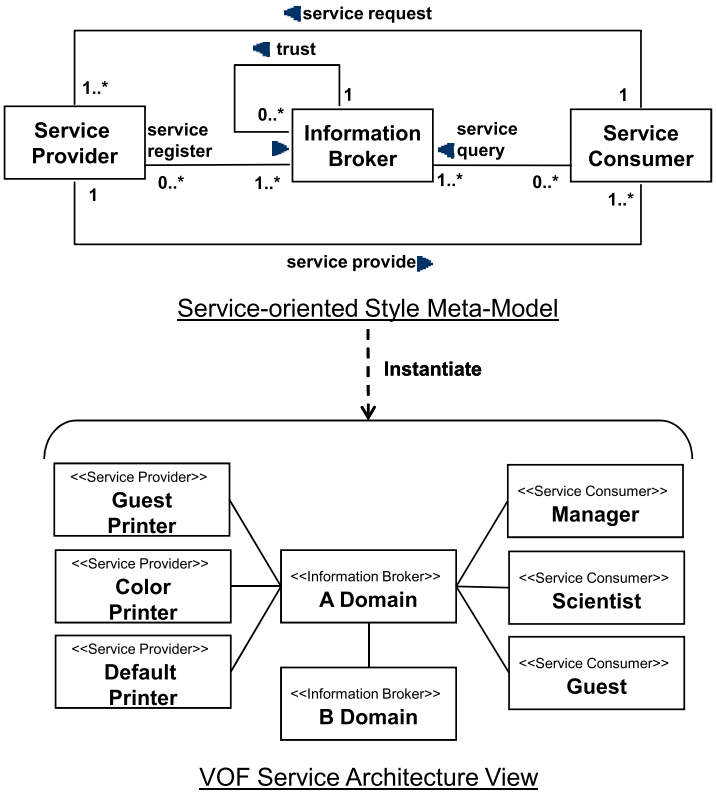

Fig. 6. A service-oriented style meta-model and its VOF instance.

The next decomposition level supports the third design goal by adapting the communicating process style. (See the upper portion of Fig. 7 for its meta-model.) The style consists of concurrent processes and their communication paths, which can be implemented independently. Obviously, each concurrent process can have its own time schedule and interact with each other via connectors. This style also benefits that the scheduling among concurrent processes can be defined in various ways (e.g., preemptability, priority, timing parameters) as needed (Clements et al., 2002).

As for its instance, we indentified four process components: *Consumer Agent*, *User Interface*, *Context Analyzer*, and *Service Manager*. (See the lower portion of Fig. 7.) The *Consumer Agent* is in charge of maintaining connectivity to the information broker and service providers. Whenever a service provider fails, it negotiates with the information broker and gets another available service provider. The *User Interface* process implements user specific hardware (e.g., PC, PDA) and operating system (e.g., Linux) relevant interfaces. The *Context Analyzer* process recognizes currently available resources (e.g., available user localization devices) and situations of a user. The *Service Manager* contains the main functionalities of a service consumer and activates relevant service features based on the information gathered from *User Interface* and *Context Analyzer*.

The next decomposition level supports the fourth, fifth, and sixth design goals by adapting C2 style[4] (Medvidovic et al., 1999). The UML based presentation of C2 style proposed in Robbins et al. (1997) is extended to include two different types of bricks: workflow brick and molecular service brick types. (The upper portion of Fig. 8 shows the adapted meta-model.) The workflow brick is for deploying

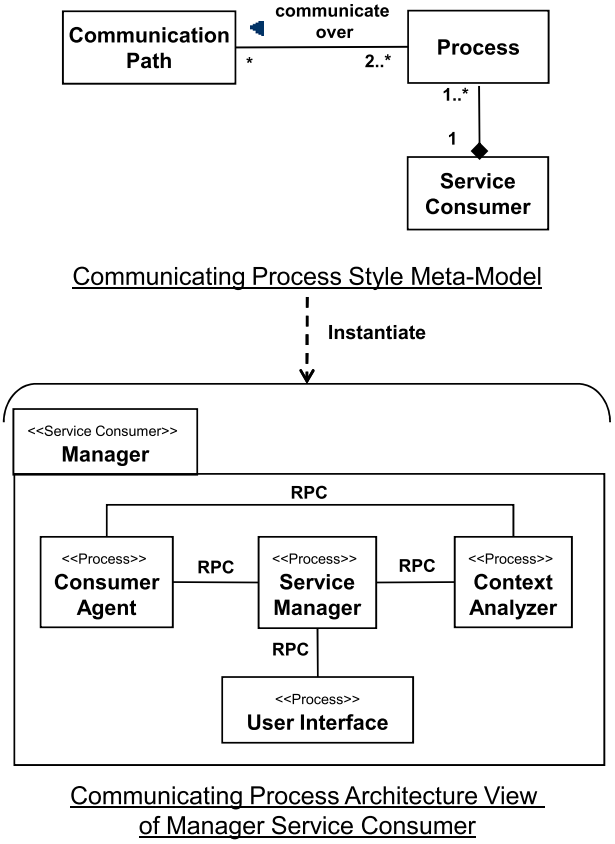

Fig. 7. A communicating process style meta-model and its VOF instance.

[4] The C2 style provides flexibility through its layered structure and modular components, which are called "bricks". A brick can have its own thread and can send/receive messages to/from other bricks through its top and bottom ports, and bus-style connectors connecting ports.

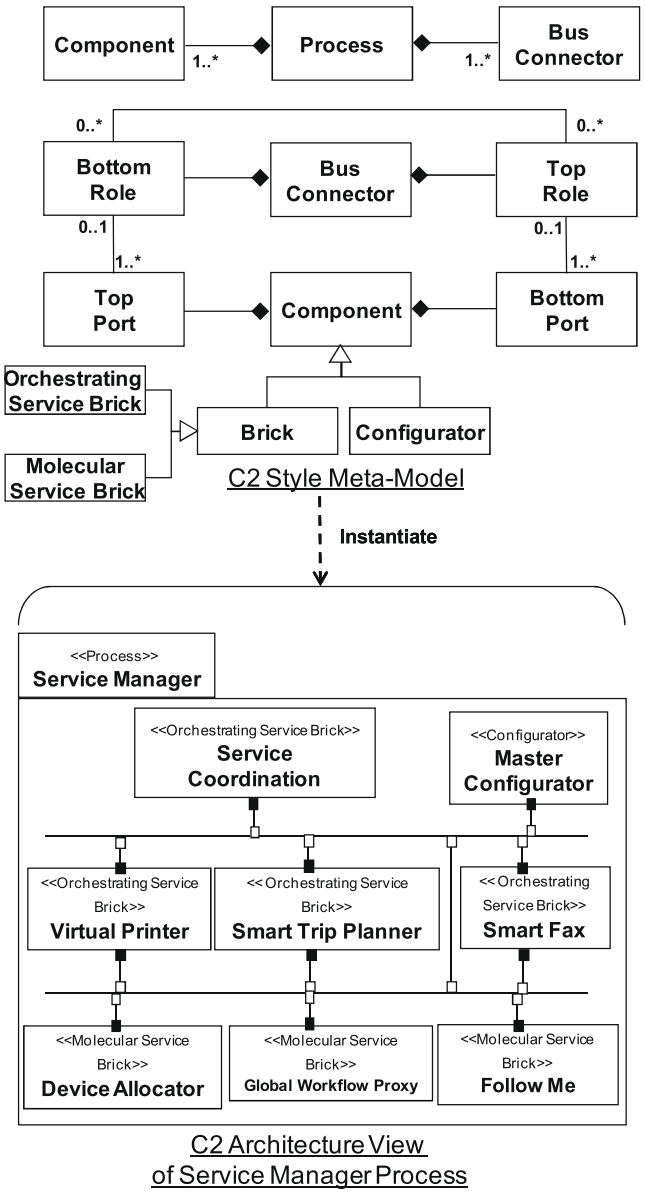

Fig. 8. A C2 Style meta-model and its VOF instance. (Adapted from Robbins et al. (1997))

orchestrating services and the molecular service brick is for deploying molecular services. For molecular services, it is possible to either deploy a real service or a proxy to an external service as a brick. Additionally, the configurator component manages reconfiguration of deployed product at runtime.

The lower portion of Fig. 8 shows a C2 style based configuration of the *Service Manager* process. The *Master Configurator* collects information from *Context Analyzer* to detect contextual changes. If a contextual change that requires product reconfiguration is detected, it triggers a reconfiguration to accommodate the change. For example, if *Context Analyzer* reports that a RFID device for the user localization is detected, it dynamically binds a corresponding *Follow Me* brick, which is capable of processing the new device. Also, when a new orchestrating service is requested and it is available, it can be deployed and bound to current configuration at runtime.

An orchestrating service brick transacts its orchestrating service locally if possible (e.g., *Virtual Printer* can be transacted locally without connecting to other users). When it requires a global coordination (e.g., an approval of deciding staff for a business trip), a global workflow engine is connected through *Global Workflow Proxy* for a global workflow transaction.

For the deployment of a molecular service brick, we can use two different strategies depending on its characteristics. If it should be dedicated to a certain user, a user specific brick is deployed locally. For instance, the *Follow Me* molecular service is dependent to a user's role and must be deployed individually. On the other hand, if it should be shared among service consumers, it is deployed as a service provider at the top level of the architecture model and a proxy brick is deployed locally. *Global Workflow Proxy* is such examples. (See the bottom layer of *Service Manager* in Fig. 8.)

In this section, we explained the HEART model for the systematic deployment and management of the system configuration with the VOF system example. In the next section, the product engineering is explained.

5. Product engineering: putting them together

Product engineering is a process of developing a specific application making use of the core assets obtained during product line asset engineering. Product engineering first analyzes user's requirements and selects appropriate and valid product features from the feature model. Then, the architecture model and components are configured for a product. Finally, a product specific service configuration is specified and allocated to the *Master Configurator*.

5.1. Product requirements analysis

As indicated above, product engineering starts with a feature selection specification. This is done by first analyzing user's requirements for the target product, and finding a matching set of features from the feature model. The feature model not only lists user selectable characteristics of the system, but also includes their interrelationships and selection criteria (e.g., rationale).

Selecting a set of features from the feature model can generate a configuration of a product. Since mandatory features will automatically be included in all products, selection should be made for optional and alternative features. In addition, some feature may need to be refined with product specific requirements. For instance, detailed behavior of the *Smart Business Trip* should be specified to meet customer specific (e.g., a guest, manager, or director) needs by using workflows.

Though some features may be physically included in a product, their availabilities still can be controlled at runtime depending on the current context. Obviously, if these features are determined not to be included in a product (e.g., a configuration for a guest user), they cannot be made available at runtime. If these features are included in a product, however, their availabilities should be determined by a user input or an operational contextual change at runtime. Therefore, a product specific context, which decides when to make these features available, should be also identified and then specified as described in Section 3.3. For example, a product for a director may make the *Automatic Log-on* service feature available automatically when its required sensors and actuators could be found through the information broker at runtime, while a product for a guest may not aware of such services due to its product configuration.

As feature selection decisions are made, the product architecture components are configured.

5.2. Configuring product architecture and components

After the feature selection process described above, a corresponding product architecture model is first configured through

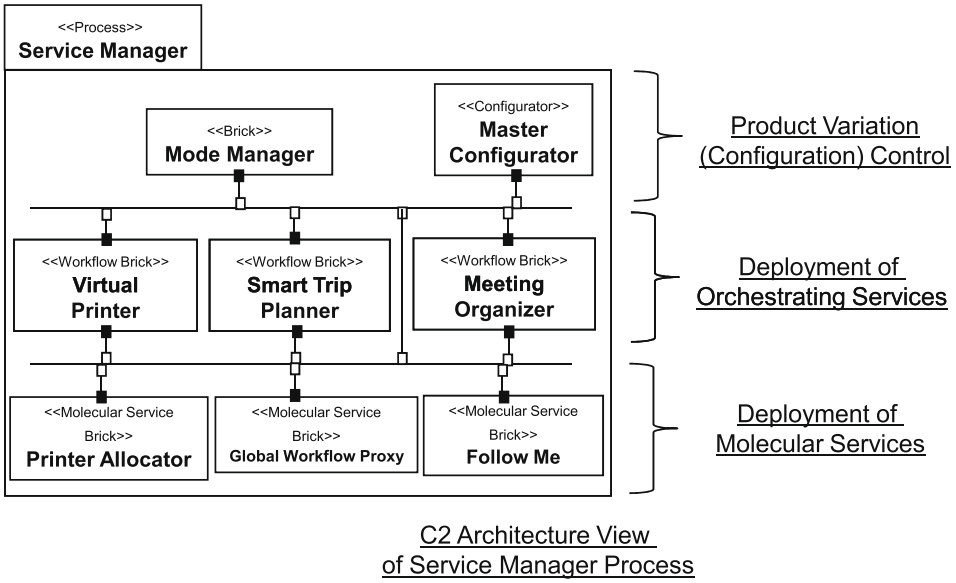

Fig. 9. Deployment of orchestrating/molecular services for a product configuration.

the mapping relation between features, services, and architectural components. Once product architecture is configured, reusable components are integrated into each bricks by following the specifications described in the components (e.g., selection of pre-coded components, filling in skeletons, or instantiation of parameterized templates).

Fig. 9 shows an example of product architecture at the last decomposition level (i.e., the C2 style), which is configured for a director. The selected components should be processed by using appropriate techniques depending on the binding time. For example, components and program segments related to the *Automatic Log-on* feature is managed within the *Follow Me* brick based on the context information gathered by the *Context Analyzer* in the upper layer (see Fig. 7) and passed through the *Master Configurator*. Fig. 9 also shows the mapping from *Orchestrating/Molecular Services* to *Workflow/Molecular Bricks*.

Note that the configuration of this layer explicitly shows a product configuration of a specific user. That is, the product variation control for a specific user is achieved by determining the deployment of bricks based on the feature selection specification for the user.

To demonstrate the feasibility of the proposed method, we developed a prototype of the VOF system. In the next section, we describe the implemented prototype in detail.

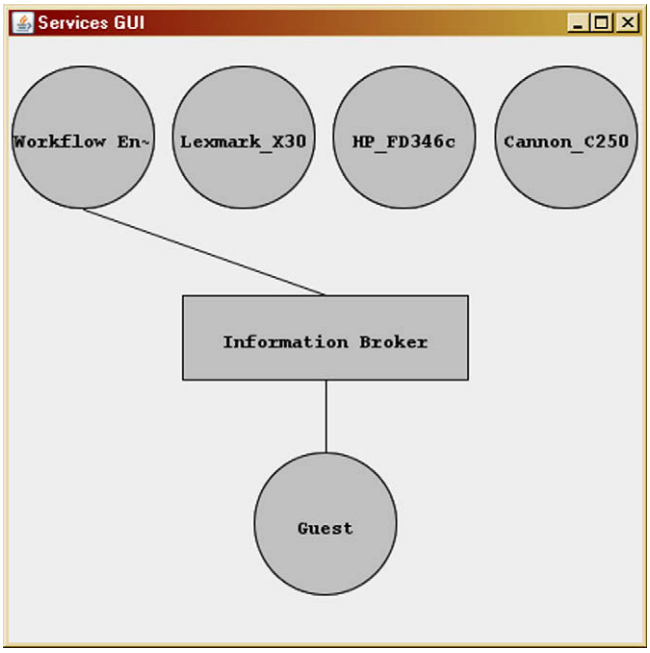

Fig. 10. Architecture visualization – service requests.

6. Case study: office system

Our case study realizes large parts of the concepts described in the previous sections. Thereby, both, the architectural concepts and the features of the office domain are realized. Additionally, the prototype was extended with an architecture visualization, which shows the nested levels and the instantiated components of the architectural styles. The main purpose of this architecture visualization is to show how messages and service requests are transported and how dynamic reconfiguration of the system according to recognized context and failure situations happens.

Fig. 10 shows the top-level architectural style, depicting service providers (respectively the services), the information broker mediating the service requests and a single service consumer, which is a client application. The lines drawn in the visualization represent the dynamic behavior, therefore they are drawn in the moment when a service request happens, after a short amount of time they disappear again. This makes it easy to represent actual situations and the particular behavior of the system.

Fig. 11 shows the internals of an application following the lower two levels of architectural styles. The main point in this part of the architecture visualization is the dynamic reconfiguration of the different types of service bricks. As described, the reconfiguration is controlled by the current context of the system. To simulate this, we simply provide a selection box, which offers a number of choices. The selection of one of the choices in the visualization is forwarded to the context manager and then leads to the reconfiguration of the services, which is visually represented by reducing the speed of the processing and highlighting the removed or introduced bricks by colored blinking.

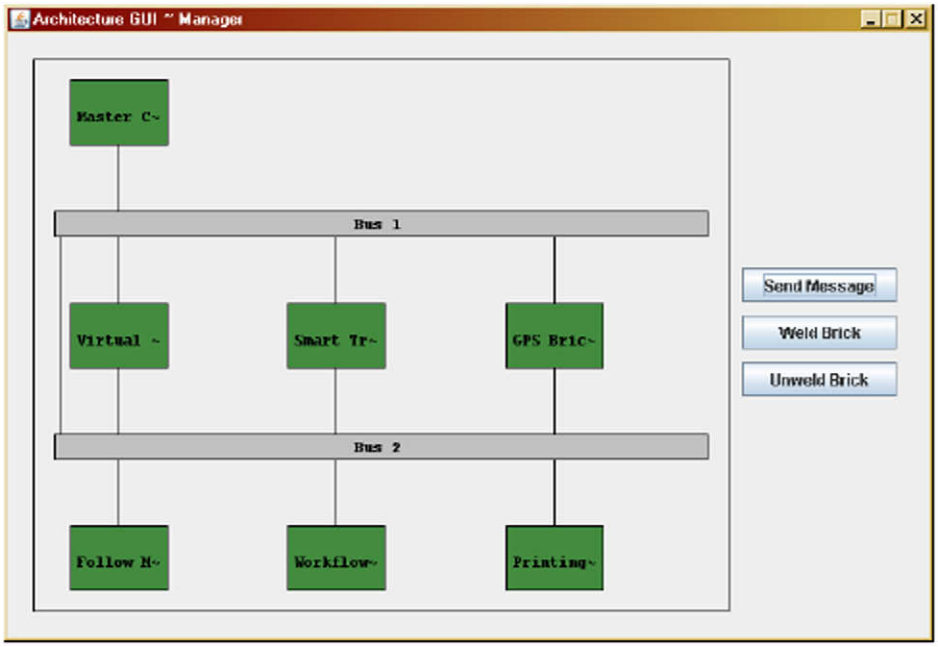

Fig. 11. Architecture visualization – service consumer's internals.

Section 6.1 describes the technical realization of the prototype, then Section 6.2 outlines the application scenarios particularly supported by the architecture and pointed out by the architectural visualization.

6.1. Technical realization

The architecture visualization is not an integral part of the architectural style. It is an external application that made some adjustments of the overall system necessary in order to gather the information necessary for the visualization. Mainly that means that notifications about events like requesting services and triggering reconfigurations are forwarded to the visualization application.

From the top level perspective, the overall system consists of at least three independent building blocks: At least one global service provider, the information broker, and at least one client application. These building blocks can be independently deployed at arbitrary places, since the communication among them is completely based on Web Services Technology for remote communication.

In the current scenario, one *service provider* is in place, which is realized as an Apache Tomcat web server. The global services themselves are realized as Web Services based on Apache Axis. Consequently, independent services can be deployed on our service provider.

The *information broker* is also provided as a Web Service. It offers the possibility to register services with simple Quality of Service (QoS) attributes. Besides the functionality of simply querying for the address of a certain service, it also provides the possibility to automatically call the service that fits the service query best. Then it returns the result to the service requesting client application.

The *client applications* are based on a common infrastructure following the two lower level architectural styles as described above. They are Java applications instantiating different threads running the GUI, the context recognition, the consumer agent and the service management.

The GUI is not to be mistaken with the GUI for the architecture visualization. The GUI of the client applications provides access to the workflows and services offered by an application. For example, workflows can be started, tasks can be processed, and the current workflow state is shown.

In the service manager component, all workflows and services (that is, the orchestrating and the molecular services) are managed in two levels of the C2 style as independent bricks. Communication among these bricks is done via messages. The direction of the communication is prescribed by the C2 style: workflows are allowed to request molecular services, but not vice versa. By means of the C2 style, the reconfiguration of the system is pretty easy. Since bricks are only loosely coupled to each other with uniform interfaces, they can be dynamically exchanged. This reconfiguration is based on information collected in the context analyzer. In the current implementation, there is no sophisticated system behind it, context information is simply simulated.

A central infrastructure component in our realization of a client application is the workflow execution system. However, it is not part of the architectural style, since workflows could be realized in other ways. Parallel to the workflow execution system at the global level, we also chose jBPM (JBoss jBPM) for workflow management at the local level of the client. This makes the exchangeability of workflow descriptions much easier. With jBPM comes the workflow description language jPDL (JBoss jBPM 2.0 jPdl).

We extended this workflow definition language with our concepts of pre-conditions, post-conditions, and invariants. Both, the overall process and the concrete building blocks, the nodes in the workflow, had to be extended. The condition can be formulated in Java code with the possibility to access all objects available in the context of the workflow. Consequently, the workflow engine itself had to be extended too, to be able to check the conditions. In case of violated conditions, different actions are possible: The user can be asked how to proceed, the workflow can be stopped and all involved users are notified, or the violation is simply logged for later analysis.

At the client site (e.g. a mobile device), the local workflow execution system can execute a workflow as long as the characteristic of locality applies, that is, no other person has to be involved. If the coordination with other persons is necessary, the global workflow execution system becomes involved by sending the control over a workflow to it. Principally, all communication is triggered by client

applications. That is, only if a client application requests the global workflow state, it becomes aware of it. To overcome that, we added beyond the style a notification message from the global workflow execution system to the local ones. Among client applications, there is principally no communication.

As described in Section 4.2, services can be deployed locally or globally. In our prototype, a print service is a candidate that can be deployed locally or globally. In the local case, a printer that is directly connected to the device, on which the application is running, can be used exclusively by the applications on that device. In the global case, a printer is a central infrastructure device, which can be remotely used by all authorized applications on distributed devices. For workflows consuming services, the location of the service provision is fully transparent. In case of globally available services, a proxy brick is configured, which externally looks identical to the real service. This proxy brick redirects the service request via the consumer agent to the external service.

The architectural principles of the client applications strongly support the idea of producing these applications in a product line. Different persons have different interests regarding their office applications. These diverging interests and needs can be expressed (at least to a certain degree) by means of adapted or different workflows and services. The architectural framework allows the easy production of applications according to a user's needs. Therefore, the workflows and services can be tailored and easily integrated.

6.2. Application scenarios

Several application scenarios were already mentioned in the previous sections. They served on the one hand as a motivation for the construction and particular design decisions. On the other hand, they were used to illustrate how a concrete system could work. In this section, we provide more substance with respect to three scenarios by explaining in detail how a scenario works and how it is supported by the architecture. These application scenarios are realized in our prototype.

6.2.1. Context-dependent reconfiguration of services

Situation: A client application is in operation and has a certain configuration of available services and workflows. The workflows and services are presented to the user via the GUI.

Trigger: The context analyzer recognizes a change in the contextual environment of the client application. Therefore, different types of sensors can be used, the data of which is processed for the recognition of context. A potential trigger is that the user enters the area of a wireless network, which can be used for localization purposes.

System response: The context analyzer generates an event that the context changed. This event is sent to the configurator of the service manager. The configurator is prepared with a set of rules that align contextual parameters and the system's response in terms of available services and workflows. Consequently, the configurator initiates that a service is deployed; this service can process information from a wireless access-point in order to calculate the device's position.

For the reconfiguration, it is important to maintain consistency properties of the client application. That is, it has to be avoided that a service is undeployed, while it is currently being used. Thus, the reconfiguration might take some time to be safe. Finally, a notification about the new configuration can be sent to all current bricks. This allows other bricks to use the services according the new configuration.

6.2.2. Distributed workflow processing with violated invariant

Situation: In a company, an automated business process for the application for leave is in place. This workflow can be provided for all employees of the company. An employee can start the workflow and then it has to be processed first by the line manager for official acceptance and then by a secretary for administrative work. In the description of the workflow, there is the invariant that at no time of the workflow processing the vacation period may conflict with official vacation bans in the company.

Trigger: An employee applies for vacation. Therefore, he uses his office application and enters all the data that is necessary to initiate the application for vacation. Then, he submits the request. Since the official acceptance has to be given by the line manager, a global coordination becomes necessary. The employee's device transfers the instance of the actual workflow over the respective proxy to the global workflow execution system.

This system recognizes that the next step has to be done by the line manager and looks up who is responsible. This might involve searching for a deputy in case the regular manager is not available. Then, a task is generated for the manager identified. In the following, this manager receives the workflow instance on its own office device and can process the next workflow step, the official acceptance. After that, the workflow instance is transferred back to the global workflow execution system. Now it is assumed that the administration of the company comes up with a vacation ban for all employees in exactly the time for which our employee has applied for vacation.

System response: The global workflow execution system is still aware of the workflow instance, since it is assigned to the secretary for finishing. Since the vacation ban leads to a violated invariant of the workflow for application for vacation, the consequence is that the workflow instance is cancelled and all involved persons are appropriately notified. That is, the employee, the manager, and the secretary are informed that the workflow was cancelled due to the vacation ban.

6.2.3. Global service call with fallback on failure

Situation: A client's application uses a global printing service. That is, the print can be initiated on the user's device and the processing is done according to the configuration of the service. Assume that there are multiple printing services available, each with different quality attributes (e.g. concerning color, resolution, etc.). These services are registered with the information broker.

Trigger: A user is processing a workflow that requires printing. Since no local printer is installed, a global print service is to be used.

System response: The request is sent via a proxy brick to the consumer agent, which contacts the information broker to get an appropriate service. Such a service is returned and the consumer agent requests the service. Assume that the service suddenly becomes unavailable and thus the consumer agent receives a time-out. Then, the consumer agent automatically sends another request to the information broker, including the information that the service retrieved before is unavailable. The information broker delivers the next best choice and the consumer agent can access this service. Finally, the result of a successfully finished printing is returned via the proxy brick to the requesting workflow. Consequently, the problems with service availability are totally transparent for service users, at least as long as an alternative service can be found.

7. Conclusion

We transfer product line technology into industry since 1998 and we experienced in nearly all cases a quick increase of the number features, as well as required variants. Hence, the management of features and their variations becomes soon one of the major challenges in maintaining and evolving viable reuse infrastruc-

tures. The environment and context of service-based systems is typically very dynamic and always distributed. Our experience with such service-based product lines has shown that the challenge of managing variations and keeping services reusable and useful over a long period of time is even bigger than for other systems.

In this paper, we presented an approach that alleviates this difficulty through the grouping of features into feature binding units of the same binding time, as well as by interpreting these units as key drivers for identifying reusable services, that is, molecular services. By clearly defining the transition from feature analysis to service analysis, we made use of the established approach of feature analysis for the systematic derivation of services. It has been shown, that sound analogies can be found that allow combining the best of both paradigms for the construction of flexible systems and product lines.

The practical applications of our approach in our lab infrastructure demonstrated that product line technology can significantly help in mastering this challenge. The key properties of the approach are its support for identifying reusable services at the right level of granularity abstraction and for deploying them at the HEART model-based system execution environment.

Currently, we are establishing a demonstration facility within our institute to execute real scenarios of a virtual office of the future. The infrastructure of this demonstration facility has been defined by following our approach, which has already provided useful conceptual insights and lessons learned from a practitioner's perspective. The case study described in this paper exemplarily describes scenarios we already support. It shows the feasibility of the overall approach. Following the approach, we constructed step-by-step the prototype.

References

Apel, S., Kaestner, C., Lengauer, C., 2008. Research challenges in the tension between features and services. In: SDSOA'08: Proceedings of the 2nd International Workshop on Systems Development in SOA Environments. ACM, New York, NY, USA, pp. 53–58.

Arsanjani, A., 2004. Service-oriented modeling and architecture – how to identify, specify, and realize services for your SOA. <http://www.ibm.com/developerworks/libr>.

Arsanjani, A., Allam, A., 2006. Service-oriented modeling and architecture for realization of an SOA. In: Proceedings of the IEEE International Conference on Services Computing. IEEE Computer Society, p. 521.

Arsanjani, A., Zhang, Liang-Jie, Ellis, M., Allam, A., Channabasavaiah, K., 2007a. S3: a service-oriented reference architecture. IT Professional 9.

Arsanjani, A., Zhang, L., Ellis, M., Allam, A., Channabasavaiah, K., 2007b. Design a SOA solution using reference architecture. IBM.

Bencomo, N., Sawyer, P., Blair, G., Grace, P., 2008. Dynamically adaptive systems are product lines too: using model-driven techniques to capture dynamic variability of adaptive systems. In: 2nd International Workshop on Dynamic Software Product Lines (DSPL 2008), Limerick, Ireland

Chen, F., Li, S., Chu, W.C.-C., 2005. Feature analysis for service-oriented reengineering. In: Proceedings of the 12th Asia-Pacific Software Engineering Conference (APSEC'05). IEEE Computer Society, pp. 201–208.

Clements, P., Northrop, L., 2002. Software Product Lines: Practices and Pattern. Addison Wesley, Upper Saddle River, NJ.

Clements, P., Bachmann, F., Bass, L., Garlan, D., Ivers, J., Little, R., Nord, R., Stafford, J., 2002. Documenting Software Architectures – Views and Beyond. Addison Wesley.

Dan, A., Johnson, R.D., Carrato, T., 2008. SOA Service Reuse by Design. In: SDSOA'08, Leipzig, Germany.

Durvasula, S. et al., 2007. SOA Practitioners' Guide Part 2: SOA Reference Architecture. <www.soablueprint.com/whitepapers/SOAPGPart2.htm>.

Erl, T., 2008. SOA: Principles of Service Design. Prentice-Hall.

Georgantas, N., Mokhtar, S.B., Bromberg, Y., et al., 2005. The amigo service architecture for the open networked home environment. In: Proceedings of 5th Working IEEE/IFIP Conference on Software Architecture (WICSA), 2005.

Hallsteinsen, S., Stav, E., Solberg, A.J., 2006. Using product line techniques to build adaptive systems. In: The Proceedings of the 10th International on Software Product Line Conference. IEEE Computer Society, Washington, DC, USA, pp. 141–150.

Helferich, A., Herzwurm, G., Jesse, S., Mikusz, M., 2007. Software product lines, service-oriented architecture and frameworks: Worlds apart or ideal partners? In: Trends in Enterprise Application Architecture. Lecture Notes in Computer Science, vol. 4473. Springer, pp. 187–201.

JBoss jBPM. <http://www.jboss.com/products/jbpm>.

JBoss jBPM 2.0 jPdl Reference Manual. <http://www.jboss.com/products/jbpm/docs/jpdl>.

Juric, M.B. et al., 2003. Business Process Execution Language for Web Services. Packt Publishing.

Kang, K., Lee, J., Donohoe, P., 2002. Feature-oriented product line engineering. IEEE Software 19 (4), 58–65.

Krafzig, D., Banke, K., Slama, D., 2005. Enterprise SOA – Service-Oriented Architecture and Best Practices. Prentice-Hall.

Lee, J., Kang, K., 2004. Feature binding analysis for product line component development. In: van der Linden, F. (Ed.), Software Product Family Engineering Lecture Notes in Computer Science, vol. 3014. Springer-Verlag, Berlin Heidelberg, pp. 266–276.

Lee, J., Kang, K., 2006. A feature-oriented approach for developing dynamically reconfigurable products in product line engineering. In: Proceedings of the 10th International Software Product Line Conference. IEEE CS Press, Los Alamitos, CA, pp. 131–140.

Lee, J., Muthig, D., 2006. Feature-oriented variability management in product line engineering. Communications of ACM (December).

Lee, J., Muthig, D., 2008. Feature-oriented analysis and specification of dynamic product reconfiguration. In: Proceedings of the 10th International Conference on Software Reuse (ICSR 2008), Beijing, China, May 25–29, 2008, pp. 154–165.

Lee, K., Kang, K., Lee, J., 2002. Concepts and guidelines of feature modeling for product line software engineering. In: Gacek, C. (Ed.), Software Reuse: Methods, Techniques, and Tools, vol. 2319. Springer-Verlag, Berlin, Heidelberg, pp. 62–77.

Lee, J., Muthig, D., Naab, M., 2008. An approach for developing service oriented product lines. In: Proceedings of the 12th International Software Product Line Conference (SPLC 2008), Limerick, Ireland, September 8–12, pp. 275–284.

Medvidovic, N., Rosenblum, D.S., Taylor, R.N., 1999. A language and environment for architecture-based software development and evolution. In: Proceedings of the 21st International Conference on Software Engineering. ACM Press, New York, NY, pp. 44–53.

Meyer, B., 1991. Design by contract. In: Meyer, B., Mandroli, D. (Eds.), Advances in Object-Oriented Software Engineering. Prentice-Hall.

Michlmayr, A. et al., 2007. Towards recovering the broken SOA triangle – a software engineering perspective. In: 2nd International Workshop on Service Oriented Software Engineering, September, Dubrovnik, Croatia.

OASIS Reference Architecture. <http://wiki.oasis-open.org/soa-rm/TheArchitecture>.

Pohl, K., Böckle, G., van der Linden, F., 2005. Software Product Line Engineering: Foundations, Principles and Techniques. Springer.

Robbins, Jason E., Redmiles, David F., Rosenblum, David S., 1997. Integrating C2 with the unified modeling language. In: Proceedings of the 1997 California Software Symposium (Irvine, CA), UCI Irvine Research Unit in Software, Irvine, CA, November 7, pp. 11–18.

International Union of Pure and Applied Chemistry, 1994. "Molecule", Compendium of Chemical Terminology Internet Edition.

Competence Center for "Virtual Office of the Future." <http://www.iese.fraunhofer.de/research/vof/vof.jsp>.

Zhu, H., 2005. Building reusable components with service-oriented architectures. In: Presented at IEEE International Conference on Information Reuse and Integration.

Jaejoon Lee is a Lecturer in the School of Computing and Communications at Lancaster University. His research interests include software product line engineering, software architecture, and service-based software engineering. Lee has a Ph.D. in Computer Science and Engineering from POSTECH in South Korea.

Dirk Muthig is Chief Platform Architect in the domain of passenger services at Lufthansa Systems, Germany. In his current role, Dirk is responsible for architecture management, which includes methodology and processes, with a strong focus on productivity and quality. Before joining Lufthansa Systems, Dirk managed the Software Development division at the Fraunhofer Institute for Experimental Software Engineering (IESE). He was with IESE for more than 10 years His research has been driven by the idea of product line engineering since the late 90s. Since then, he is an active member of the product line community: Dirk was the General Chair of the Software Product Line Conference (SPLC) in 2010, as well as has taught product line and architecture course at university for several years. Dirk Muthig holds a master's degree, as well as a Ph.D., in computer science, which he both received from the Technical University of Kaiserslautern.

Matthias Naab is a researcher at the Fraunhofer Institute for Experimental Software Engineering (IESE). His main research interests are the in areas of software architecture for large-scale information systems and software product lines. The main focus of his work is on applied research and technology transfer into industry, covering various application domains. Matthias Naab received a diploma in computer science from the Technical University of Kaiserslautern in 2005.