An Approach for Developing Service Oriented Product Lines

Jaejoon Lee
Computing Department, InfoLab21, South Drive,
Lancaster University, Lancaster, United Kingdom
(tel)+44-1524-510359, (fax)+44-1524-510492
j.lee@comp.lancs.ac.uk

Dirk Muthig and Matthias Naab
Fraunhofer Institute for Experimental Software Engineering (IESE),
Fraunhofer Platz 1, 67663 Kaiserslautern, Germany
(tel) +49-631-6800-1302, (fax)+49-631-6800-1399
{dirk.muthig, matthias.naab}@iese.fraunhofer.de

ABSTRACT

Service Orientation (SO) is a relevant promising candidate for accommodating rapidly changing user needs and expectations. Adopting SO in practice for real software and system development, however, has uncovered several challenging issues, such as how to identify services, determining configurations of services that are relevant to users' current context, and maintaining system integrity after configuration changes. In this paper, we propose a method that addresses these issues by adapting a feature-oriented product line engineering approach. Our method is based on the feature analysis technique that enables us to identify services of a service oriented system. The method is notable in that it guides developers to identify services at the right level of granularity, to map users' context to relevant service configuration, and to maintain system integrity in terms of invariants and pre/post conditions of services. We also 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 [1][2][3]. This is a promising candidate for supporting continuously changing user needs and expectations, as more and more software systems are connected to the Internet. That is, their evolution could be supported and accelerated by dynamically adding and integrating services through the Internet.

Hence SO promises similar capabilities as product lines do at a first glance and it seems to require less investment because it is "only" a new technology. Adopting SO in practice for real software and system development, however, has uncovered several challenging issues, such as how to identify services, determining configurations of services that are relevant to users' current context, and maintaining system integrity after configuration changes.

In this paper, we therefore propose a method that addresses these issues by adapting a feature-oriented product line engineering approach, which has been applied successfully for establishing software reuse in practice [4][5]. The method is the novel fusion of the two research themes of services and software product line engineering: achieving flexibility of network based systems though service orientation, but still managing product variations thought product line engineering techniques. It is based on the feature analysis technique [6][7] that enables us to identify services of a service oriented system. The service features may vary from a user's point of view and thus will be subjects of configuration changes of service oriented systems. At the same time, the method preserves the main characteristic of service orientation: a network of services that might not be designed to work together but must work together to satisfy user-specific needs. To this end, some service features are selected and/or parameterized at runtime by a user or by a product itself when a certain contextual change or a new service provider is recognized.

The method is notable in that it guides developers to identify services at the right level of granularity, to map users' context to relevant service configuration, and to maintain system integrity in terms of invariants and pre/post conditions of services. We also propose a heterogeneous style based architecture model for a systematic development of such systems.

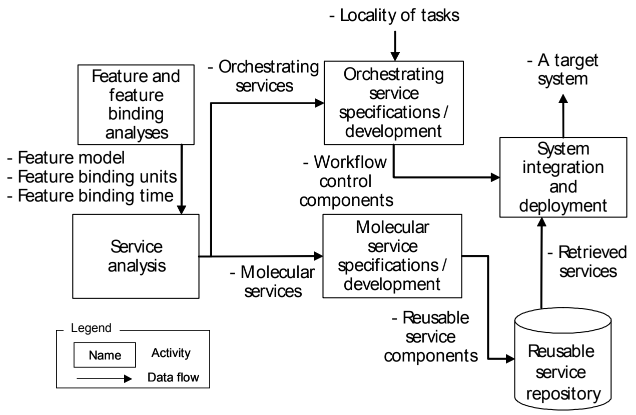
The remainder of this paper is organized as follows: Section 2 gives an overview of our approach, as well as introduces the case study used to illustrate it throughout the paper. Section 3 and 4 present the two main steps of the approach, that is on the one hand, the identification of features and their bindings, and on the other hand, the specification of service orchestrations. Section 5 proposes an architecture model with its constituting meta-models and a running example. Section 6 discusses related works and Section 7 concludes the paper.

978-0-7695-3303-2/08 $25.00 © 2008 IEEE
DOI 10.1109/SPLC.2008.34

2. Approach Overview

Product line engineering, in general, systematically exploits common characteristics and predicted variations among products of the same family [8][9][10]. The key idea is to split the overall lifecycle into two main phases: family and application engineering. 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. Each instance of application engineering constructs and maintains one particular product.

The method guides us to construct, on the one hand, a reuse infrastructure that consists of generic services optimized for the particular family of envisioned products. On the other hand, it also guides the construction and evolution of family members, which heavily reuse existing services, as well as systematically exploit thereby the flexibility and scaleability provided by the SO paradigm.



- Locality of tasks
Feature and feature binding analyses
- Orchestrating services
Orchestrating service specifications / development
- A target system
- Feature model
- Feature binding units
- Feature binding time
- Workflow control components
System integration and deployment
Service analysis
Molecular service specifications / development
- Molecular services
- Retrieved services
- Reusable service components
Reusable service repository
Legend
Name
Activity
Data flow

Figure 1 Activities of the approach

Figure 1 shows activities and their relationships of the technical component presented. These activities are executed iteratively; the arrows in Figure 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 [11].

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 that the behaviors of services can be described best by workflows executed by the system users. Additionally, the context and the available technical infrastructures may vary and thus dynamic reconfigurations of product variants are expected.

The mass of low level services, that we call atomic services, are grouped into richer services as required by the family. These richer services are (virtually) composed of atomic services and thus we call them as 'molecular[1]' services. Note that each product family has thus its own specific set of molecules, the basic building blocks for constructing family members. Due to the definition of those molecules based on product line processes, molecular services are more reusable than atomic 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.

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 [12]. In this paper, we limit ourselves to the following VOF features:

- ***Follow Me***: 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, RFID, etc. 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) more than one locating device is available nearby.
- ***Resource Manager***: A major role of a resource manager is to keep track of availabilities of office peripherals (e.g., printers, fax machines, scanners,

[1] 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' [13]. 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.

etc.). Their availabilities may change over time, as some devices are newly installed and some are removed or turned off for maintenance. In addition, the resource manager should keep attributes of each device such as its physical location and capabilities (e.g., supported paper sizes of a printer). When a user requests devices that are required for her/his tasks, the resource manager allocates available devices to the user based on a pre-determined strategy (e.g., shortest-distance- or attribute-based strategy).

- ***Virtual Printer***: 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***: 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.

3. Feature Analysis

In this section, activities of feature analysis, which includes feature modeling and feature binding analysis are introduced. 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 [7]. 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.

Common features among different products in a product line are modeled as mandatory features (e.g., *Resource Manager* and *Follow Me*), while different features among them may be optional (e.g., *Automatic Log-on*) or alternative (e.g., *AP-based* or *RFID-based User Localizer*). 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. Details of feature analysis and guidelines can be found in [7].

Once we have a feature model, it is further analyzed through feature binding analysis [11]. 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, *Follow Me*, *Resource Management*, *Virtual Printer*, 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. By taking these feature binding units as a key driver for service analysis, we could alleviate the difficulties for identifying candidate services with right granularity, i.e., reusable services.

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

4. 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 a separation of two distinctive service characteristics: behavioral (workflow) and computational (tasks) service characteristics.

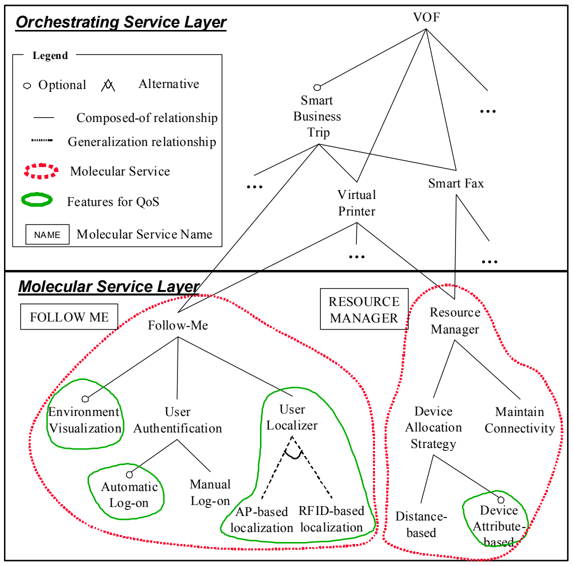


Figure 2 A Refined Feature Model based on Two Service Categories

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 predefined 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 Figure 2 for the refined feature model with the two service layers.)

4.1 Orchestrating Service

For orchestrating services, correctness of their overall control behavior is the foremost concern. For example, providing an expensive color-printing service with proper authorization and billing processes is critical for virtual office service providers. Therefore, adopting a formal method framework to specify, validate, and verify is the most suitable way for developing orchestrating services. In our approach, we adapted a workflow specification language [14] with pre/post conditions and invariants to enhance the reliability of specifications.

Figure 3 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 Figure 3.) Whenever the invariants become invalid, the workflow is terminated with proper notifications to relevant stakeholders.

Moreover, each task of the workflow can be specified with its pre/post conditions and invariants. 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 note that the consistency of invariants between a workflow and its constituting tasks should be checked when an orchestrating service is specified.

In addition to the identification of tasks and their pre/post conditions and invariants for an orchestrating service, the locality of each task should also be identified for high availability of services. 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; that is there is no switch of control to other persons in between.

The locality information is particularly important for a domain that should support mobility of users like the VOF systems. For instance, the visa process and reservation tasks are local to a secretary and they can be processed without the coordination at the global level. This means that the secretary can perform the tasks locally though she/he is disconnected from a network. However, the transaction between the approval task by a deciding staff and the visa process task by a secretary should be managed at the global level, because they belong to two different persons.

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

4.2 Molecular Services

The identification of molecular services with right

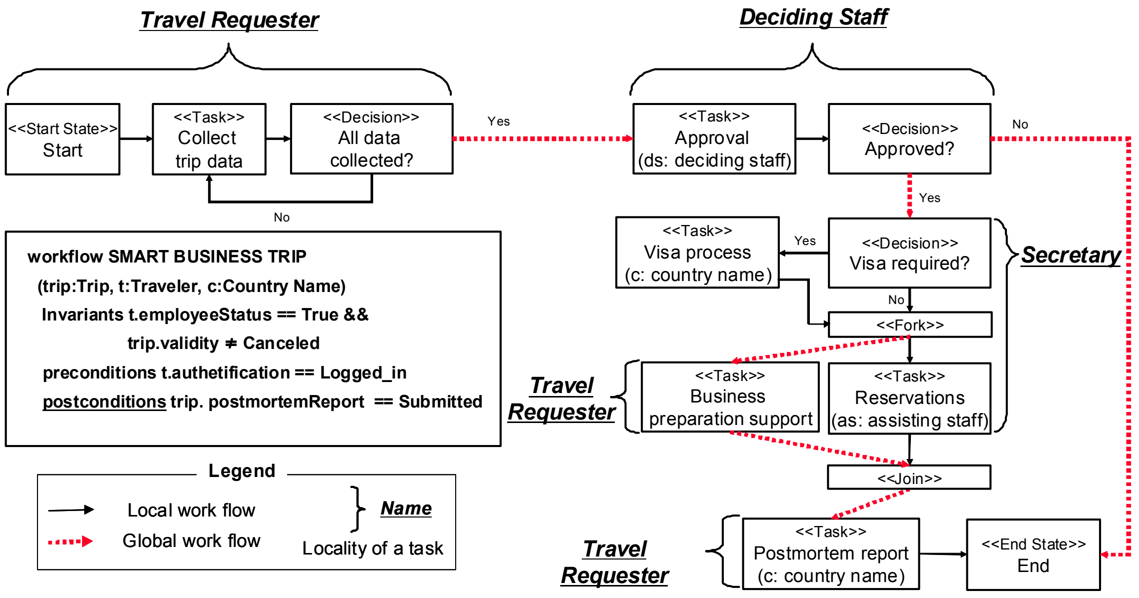

Travel Requester
Deciding Staff
<<Start State>> Start
<<Task>> Collect trip data
<<Decision>> All data collected?
No
Yes
<<Task>> Approval (ds: deciding staff)
<<Decision>> Approved?
No
Yes
<<Task>> Visa process (c: country name)
<<Decision>> Visa required?
Secretary
No
<<Fork>>
Travel Requester
<<Task>> Business preparation support
<<Task>> Reservations (as: assisting staff)
<<Join>>
Travel Requester
<<Task>> Postmortem report (c: country name)
<<End State>> End
workflow SMART BUSINESS TRIP
(trip:Trip, t:Traveler, c:Country Name)
Invariants t.employeeStatus == True &&
trip.validity ≠ Canceled
preconditions t.authetification == Logged_in
postconditions trip. postmortemReport == Submitted
Legend
Local work flow
Global work flow
Name
Locality of a task

Figure 3 An Example of Workflow Specification for an Orchestrating Service: Smart Business Trip

granularity is the key factor to enhance reusability of the service oriented 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. 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 must not be propagated to other services.

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. 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 Figure 2. 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*.

Each molecular service may have its QoS parameters, which are identified during the feature binding analysis in terms of optional or alternative features. For example, the *User Localizer* feature has two alternatives (e.g., *AP-based localization* and *RFID-based localization*) 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 meters.). Depending on available devices near a user, one of the alternative positioning methods is selected and used.

In our approach, each molecular service is specified by using a text-based specification template and

Figure 4 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 to 13 for the specification of optional feature *Automatic Log-on*.)

```
 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;
```

Figure 4 An Example of Molecular Service Specification

In this section, concepts and guidelines for analyzing and specifying molecular services are explained. The next section introduces an architecture model for the development of these services.

5. A Heterogeneous Style based Architecture Model

For the development of the VOF system, we propose a **he**terogeneous style based **ar**chi**t**ecture (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.

5.1 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 resources to their consumers: this is to achieve the runtime flexibility, which is the main idea of SO. By networked resources 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 have to be supported by mobile devices that 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 resource consumer: a resource 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.

In the following section, we explain our proposed architecture model and how these design goals are achieved.

5.2 The HEART Model

The HEART model consists of three decomposition levels and each level addresses specific design goals listed above by adopting architecture styles [15]. The top level supports the fist and second design goals by adopting the service oriented style. (Figure 5 shows the meta-model of the style.) Resource 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 resource consumers can join the trusted domain and access resources in that domain.

Note that we did not impose any constrains on resource providers, as long as they abide by the interfaces with the information broker and service consumers.

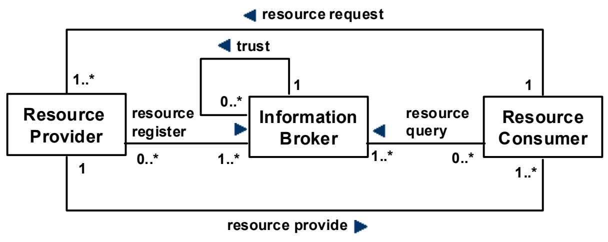


Figure 5 A meta-model of the top decomposition level of HEART: A Service-Oriented Style

The next decomposition level supports the third design goal by adapting the communicating process style. (See Figure 6 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 [15].

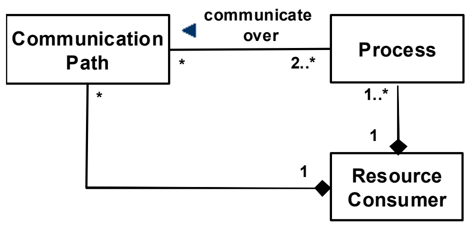


Figure 6 A meta-model of the second decomposition level of HEART: A Communicating Process Style

The next decomposition level supports the fourth and fifth design goals by adapting C2 style[2] [16]. The UML based presentation of C2 style proposed in [17] is extended to include two different types of bricks: workflow brick and molecular service brick types. (Figure 7 shows the adapted meta-model.) The workflow brick is for deploying 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.

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

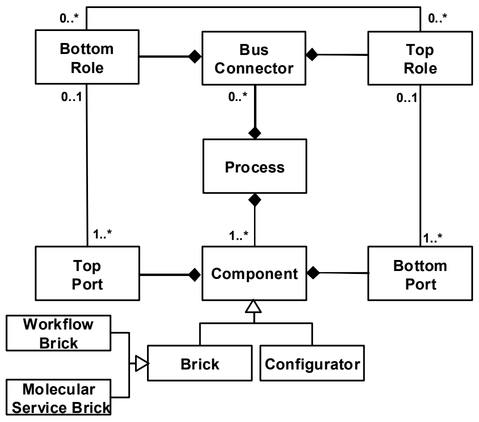


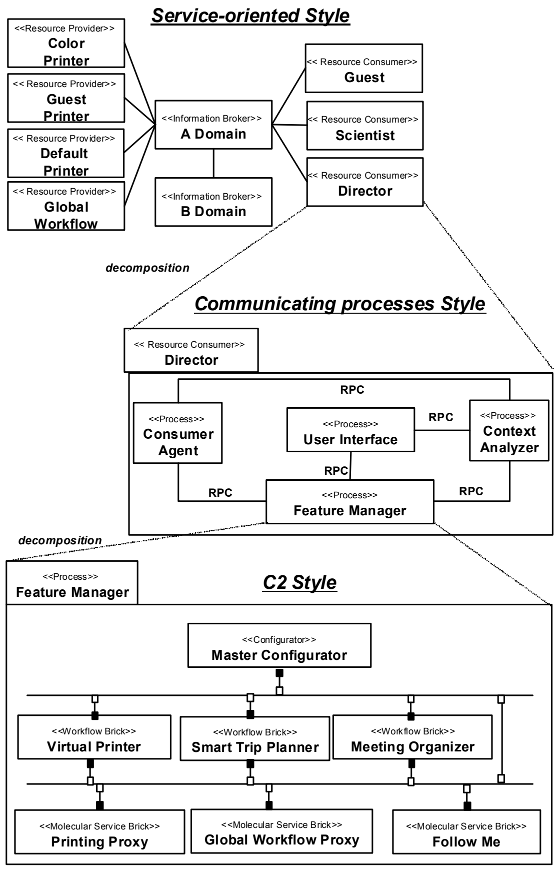
Figure 7 A meta-model of the lowest decomposition level of HEART: A C2 Style (Adapted from [17])

5.3 An Instance of HEART: the VOF System

Figure 8 shows a deployed VOF system, which is an instance of the HEART model. At the top level, three printing resource providers are deployed: *Guest Printer*, *Color Printer*, and *Default Printer*. Each resource 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 resource consumers are deployed: *Director*, *Scientist*, and *Guest*. Each name represents its role and its accessibility to the resource providers are decided by the role. For example, *Director* can access all printer resources, while *Guest* can only access the *Guest Printer* resource provider. This information is maintained by the information broker.

At the next level, we indentified four process components: *Consumer Agent*, *User Interface*, *Context Analyzer*, and *Feature Manager*. The *Consumer Agent* is in charge of maintaining connectivity to the information broker and resource providers. Whenever a resource provider fails, it negotiates with the information broker and gets another available resource 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 devices for a user. The *Feature Manager* contains the main functionalities of a resource consumer and activates relevant features based on the information gathered from *User Interface* and *Context Analyzer*.

The lowest level shows a C2 style based configuration of the *Feature 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.



Service-oriented Style
<<Resource Provider>> Color Printer
<<Resource Provider>> Guest Printer
<<Resource Provider>> Default Printer
<<Resource Provider>> Global Workflow
<<Information Broker>> A Domain
<<Information Broker>> B Domain
<< Resource Consumer>> Guest
<< Resource Consumer>> Scientist
<< Resource Consumer>> Director
decomposition
Communicating processes Style
<< Resource Consumer>> Director
RPC
<<Process>> Consumer Agent
<<Process>> User Interface
<<Process>> Context Analyzer
RPC
RPC
RPC
RPC
<<Process>> Feature Manager
decomposition
<<Process>> Feature Manager
C2 Style
<<Configurator>> Master Configurator
<<Workflow Brick>> Virtual Printer
<<Workflow Brick>> Smart Trip Planner
<<Workflow Brick>> Meeting Organizer
<<Molecular Service Brick>> Printing Proxy
<<Molecular Service Brick>> Global Workflow Proxy
<<Molecular Service Brick>> Follow Me

Figure 8 An instance of the HEART model: a deployed VOF system

The workflow 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 resource provider at the top level of the architecture model and a proxy is deployed locally. *Printing* and *Global Work-*

flow are such examples and only their proxies are deployed locally. (See the bottom layer of Figure 8.)

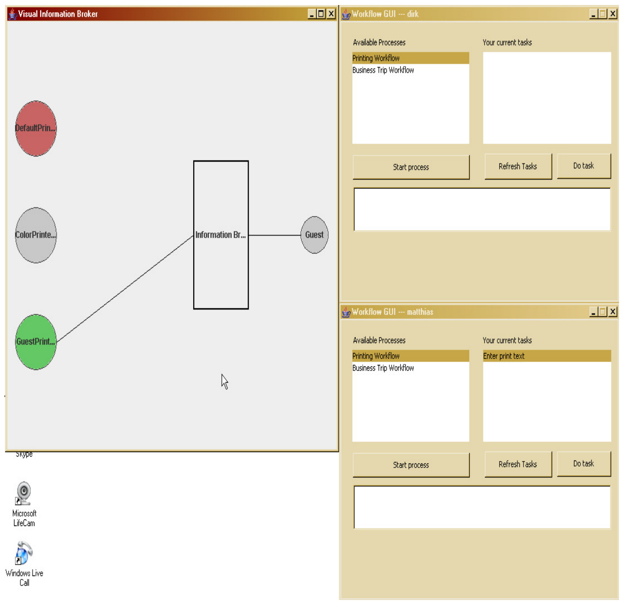


Figure 9 A Screen Capture of the Prototype of VOF

In this section, we explained the HEART model for the systematic deployment and management of the system configuration with the VOF system example. To demonstrate the feasibility of the proposed method, we developed a prototype of the VOF system and Figure 9 is a screen capture of the system demonstration. The left portion of Figure 9 shows the first layer of HEART: when a printer resource becomes unavailable, it is marked as unavailable resource (the red circle) and another available printer (the greed circle) is allocated to the resource consumer. The right portion of Figure 9 shows the GUI of two resources consumers: it shows the current available services and, if the user invoked any services, their active tasks. For example, the second (guest) user invoked the virtual printer service and the *Enter Print Text* task is shown at the *Your Current Task* window. In the following, we discuss related works.

6. Related Work

While the approach in this paper concentrates on achieving reusability by means of proper identification and specification of services using product line technologies, [2] 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.

IBM developed a method for the development of SO systems called 'Service-Oriented Modeling and Architecture' [3][18]. It provides guidelines for three steps towards SO systems: Identification, specification, and realization of services, flows, and components. Most details in [3] 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 [3], 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.

Another approach of using feature-oriented analysis to identify services for a SO system is described in [19]. 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's 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) [20] 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' [21]. 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 [22][23][24][25][26]. 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 [27]. 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.

While the aforementioned reference architectures for service-oriented systems are not dedicated a specific domain, our architectural model was explicitly designed for the office domain. Thus, more specific requirements can be incorporated into the architecture model. Important concerns in the office domain are: Systems have to work independently of centralized solutions in mobile contexts, that is, a user carries his device with him and wants to work without a network connection to central servers. Further, the recognition of the current context of the user is supposed to influence the behavior of the system in a sense that always the best possible service quality is provided. This requires an architecture that allows for a flexible reconfiguration of the client system.

In order to achieve the specific requirements of the targeted domain, we focused on the architecture of resource consumers that can be deployed on independent devices. For the overall architecture of service-oriented systems, our approach can be complemented with the reference architectures found in literature. We introduced a light-weight approach for the transparent deployment of molecular services on either client or server side. Combined with the context-awareness and the runtime-flexibility, the client architecture is well-suited for the domain-specific requirements.

The description of reference architectures for service-oriented systems is often vague and ambiguous compared to state-of-the-art in architecture documentation [9]. That is, it is not clear what component and connector types are used and how the resulting system could look like. We composed our reference architecture of well-known architectural styles in a hierarchical way. Thus, the documentation could be clear and unambiguous.

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 infrastructures. The environment and context of service-oriented systems is typically very dynamic and always distributed. Our experience with such service-oriented 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.

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.

8. References

[1] http://en.wikipedia.org/wiki/Service-orientation

[2] H. Zhu, "Building reusable components with service-oriented architectures," presented at IEEE International Conference on Information Reuse and Integration, (2005)

[3] A. Arsanjani, "Service-oriented modeling and architecture - How to identify, specify, and realize services for your SOA," http://www.ibm.com/developerworks/library/ws-soa-design1/ (2004)

[4] K. Kang, J. Lee, and P. Donohoe, Feature-Oriented Product Line Engineering, *IEEE Software*, 19(4), July/August (2002) 58-65

[5] J. Lee and K. Kang. A Feature-Oriented Approach for Developing Dynamically Reconfigurable Products in Product Line Engineering, *Proceedings of the 10th International Software Product Line Conference,* IEEE CS Press, Los Alamitos, CA (2006) 131-140

[6] J. Lee and D. Muthig, Feature-Oriented Variability Management in Product Line Engineering, *Communications of ACM*, December (2006)

[7] K. Lee, K. Kang, and J. Lee, Concepts and Guidelines of Feature Modeling for Product Line Software Engineering. In: Gacek, C. (eds.): Software Reuse: Methods, Techniques, and Tools. *Lecture Notes in Computer Science*, Vol. 2319. Springer-Verlag, Berlin Heidelberg (2002) 62-77

[8] J. Bayer, et al, PuLSE: A Methodology to Develop Software Product Lines. *Proceedings of the Fifth Symposium on Software Reusability. SSR'99*, ACM Press (1999)

[9] P. Clements and L. Northrop, *Software Product Lines: Practices and Pattern,* Addison Wesley, Upper Saddle River, NJ (2002)

[10] D.M. Weiss and C.T.R Lai, *Software Product-Line Engineering: A Family-Based Software Development Process*, Reading, MA: Addison Wesley Longman, Inc., (1999)

[11] J. Lee and K. Kang. Feature Binding Analysis for Product Line Component Development. In: van der Linden, F. (eds.): Software Product Family Engineering. *Lecture Notes in Computer Science*, Vol. 3014. Springer-Verlag, Berlin Heidelberg (2004) 266-276

[12] Competence Center for "Virtual Office of the Future," http://www.ricoh.rlp-labs.de/index.html

[13] International Union of Pure and Applied Chemistry (1994), "molecule", *Compendium of Chemical Terminology* Internet edition.

[14] JBoss jBPM 2.0 jPdl Reference Manual, http://www.jboss.com/products/jbpm/docs/jpdl

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

[16] N. Medvidovic, D.S. Rosenblum, R.N. Taylor, A Language and Environment for Architecture-Based Software Development and Evolution. *Proceedings of the 21st International Conference on Software Engineering*, ACM Press: New York, NY (1999) 44-53

[17] Jason E. Robbins, David F. Redmiles, and David S. Rosenblum, "Integrating C2 with the Unified Modeling Language," Proceedings of the 1997 California Software Symposium (Irvine, CA), UCI Irvine Research Unit in Software, Irvine, CA, November 7, 1997, pp. 11-18

[18] A. Arsanjani and A. Allam, "Service-Oriented Modeling and Architecture for Realization of an SOA," in Proceedings of the IEEE International Conference on Services Computing: IEEE Computer Society, (2006) 521

[19] F. Chen, S. Li, and W. C.-C. Chu, "Feature Analysis for Service-Oriented Reengineering," in Proceedings of the 12th Asia-Pacific Software Engineering Conference (APSEC'05) - Volume 00: IEEE Computer Society, (2005) 201-208

[20] Andrews, Curbera, Dholakia, Goldand, Klein, Leymann, Liu, Roller, Smith, Thatte, Trickovic, and Weerawarana, "Business Process Execution Language for Web Services," (2003)

[21] B. Meyer, "Design by Contract," in Advances in Object-Oriented Software Engineering, D. Mandroli and B. Meyer, Eds.: Prentice Hall, 1991

[22] N. Georgantas, S.B. Mokhtar, Y. Bromberg, et al., The Amigo Service Architecture for the Open Networked Home Environment. In Proceedings of 5th Working IEEE/IFIP Conference on Software Architecture, 2005. WICSA 2005.

[23] Arsanjani, A. Liang-Jie Zhang Ellis, M. Allam, A. Channabasavaiah, K. "S3: A Service-Oriented Reference Architecture". IT Professional, Vol. 9 (2007).

[24] Arsanjani, A. Liang-Jie Zhang Ellis, M. Allam, A. Channabasavaiah, K. "Design a SOA solution using reference architecture". IBM (2007).

[25] OASIS Reference Architecture, http://wiki.oasis-open.org/soa-rm/TheArchitecture

[26] S. Durvasula, et al., SOA Practitioners' Guide Part 2: SOA Reference Architecture www.soablueprint.com/whitepapers/SOAPGPart2.htm

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