Granularity of Services
An economic analysis

Bernd Heinrich, Alexander Krammer, Matthias Henneberger, Florian Lautenbacher

The choice of an adequate granularity of services also represents an economic problem. Realizing the functions of a process by implementing a large number of fine-grained services leads to reduced development and maintenance efforts. Additionally, a higher reuse potential of the services can be achieved. However, at the same time the composition costs of the (many) services increase. The paper deals with this granularity issue. For this purpose, three different metrics to measure granularity will be introduced. Furthermore an economic model for service granularity optimization will be introduced, which extends existing approaches focusing purely on a domain analysis.

Service-oriented architectures are widely discussed as a design principle for application and enterprise architectures. Nevertheless, an adequate granularity of services from an economic perspective has not yet been researched sufficiently. The finer the granularity to realize the functions of a process, the higher is the number of services and the more effort has to be spent to compose them. In contrast, very coarse-grained services bear the disadvantages of higher implementation costs and lower reuse potential (e.g., in different processes). The decision model proposed in this paper is to determine an adequate granularity of services from an economical perspective. Thus, degrees of freedom, which often exist for the choice of granularity after a domain analysis, can be leveraged to realize a cost-efficient solution. We illustrate the applicability and practical benefits of the decision model with an example in the context of a financial services provider.

Service-oriented architecture, Granularity, Metrics, Value-based software engineering

1 Problem Definition

Although service-oriented architectures (SOA) have been discussed for quite some time now, the question of an appropriate granularity of services in economic terms has not yet been answered satisfactorily. While it is often required that services should be more coarse-grained than objects or components and at the same time should be designed from a functional point of view (Henning 2007; Krafzig et al. 2005; Richter et al. 2005), such statements offer a large individual freedom regarding design alternatives. This is critical since the significance of the granularity question is emphasized in many contributions (see previous sources) and is sometimes even labeled as the crucial question (Melzer et al. 2007, S. 33).

The advantages and disadvantages of coarse- resp. fine-grained services can basically be balanced as follows (cf. Aier 2006; Erl 2005, S. 557; Melzer et al. 2007, S. 33): The more fine-grained the service, the higher the number of services to realize the functions of a process and the more effort has to be spent to compose the (multitude of) services for process execution. In contrast, coarse-grained services bear, for example, the disadvantage that the potential for reusing services in various processes decreases (cf. also Joachim et al. 2011,

S. 450). This poses the risk of redundancy and a high number of variants, as several services may realize the same or similar functions of different processes. Although services are still characterized by platform independence, usage of standards, loose coupling, etc. (Buhl et al. 2008; Papazoglou 2003; Papazoglou et al. 2006), a significant advantage vanishes: If, in extreme cases, services have a similar (coarse) granularity as monolithic application systems, the often stated advantage of modular software artifacts that can "simply" be reused and composed to new or modified processes is lost.
So far the question of service granularity has been discussed in literature primarily from a functional view (e.g., Albani et al. 2008; Fiege 2009; Winkler 2007; Winter 2003). However, this question also describes an economic problem: How granular should services be developed so that the effort for the development, composition, and maintenance of services is minimal? Such an economic approach requires appropriate metrics in order to be able to measure granularity in a comprehensible way. This paper aims at contributing to both issues. Therefore, it follows the growing insight that economic aspects for system and service design (cf. Value-based Software-Engineering; e.g., Biffl et al. 2005) have to be considered to a stronger extent than before.
The paper is organized as follows: In section 2 contributions will be discussed dealing with the identification and design of services (and components). These approaches apply primarily functional or technical criteria and thus constitute the starting point for a subsequent economic analysis. In section 3, various metrics to measure service granularity are defined. These metrics are prerequisites to make comprehensible statements, such as „realization through fine- or coarse-grained services“. Afterwards, a model is developed which supports the decision on the granularity of services based on economic criteria. The applicability of the model will be demonstrated in section 4 through a case study of a financial service provider. Section 5 summarizes the essential findings, analyzes them critically, and provides an outlook on further research needs.

2 Literature review

A series of contributions about the identification and design of services, especially based on functional criteria, can be found in the literature. Services can be understood as (software) artifacts of a system landscape which encapsulate functions (Schelp and Winter 2008, p. 6) and exhibit specific characteristics, such as modularity, loose coupling, and defined interfaces (Krafzig et al. 2005, p. 59; cf. e.g., Buhl et al. 2008, p.62; Erl 2005, p. 37 for the characteristics of services). Often, a distinction is made between technical and business (or enterprise) services, whereas the latter realize composable functions of a business process (cf. Melzer 2007, p. 32; Schelp and Winter 2008, p. 7). In the following, we restrict ourselves to the latter as the question of reusability of business services in different processes is of particular interest. Therefore the question of a “proper” granularity is raised here in particular. First, we define granularity, according to the literature, as the number or extent of functionalities implemented by a service (Erl 2007; Galster and Bucherer 2008, p. 400; Thomas et al. 2010, p. 366). Papazoglou and van den Heuvel (2006, p. 423) write for instance: “Service granularity refers to the scope of functionality exposed by a service”. According to Boerner and Goeken (2009) granularity also describes the functional scope of a service. This short discussion of the term granularity will be resumed in section 3.2 when we define granularity metrics.

For our study, work dealing with the identification of components is relevant besides those contributions dealing with the identification and design of services (for a review of approaches for component identification cf. Birkmeier and Overhage 2009; for service identification cf. Birkmeier et al. 2008). **Table 1** illustrates selected contributions of both domains, based partly on the description of Birkmeier and Overhage (2009) as well as Birkmeier et al. (2008). Subsequently, the contributions on component identification will be discussed followed by approaches on service identification.

Table 1 Selected approaches for the identification of components and services

Authors	Objective	Method	Subject	Definition of Granularity	Guidelines defining the granularity of components resp. services	Focus	Tool Support	Type of Validation
Aier 2006	Identification of services as an aggregation of associated elements in an enterprise architecture.	Automated clustering method analyses relationships between the elements of an event-driven process chain	Services	Number of the realized functions implemented by a service (implicit)	Granularity depends on determining the parameter values of the clustering method (no optimization)	Functional	Yes	Illustration example
Albani et al. 2008	Identification of business components which belong together from a functional perspective.	Refinement of domain models to components with the help of heuristics (top down)	Components	No explicit definition of granularity	Preferences to decomposition and grouping rules	Functional	Yes	Real world case study
Arsanjani et al. 2008	Development of a service-oriented architecture, taking the complete service life cycle into account.	Top down analysis of the domain and business goals (focus) with an additional bottom-up analysis of existing systems.	Services	No explicit definition of granularity	No guidelines	Functional	No	Presentation of case studies and collected best practices
Beverungen et al. 2008	Identification of services based on models of business processes.	Decomposition of business processes, taking into account visibility issues regarding business partners (top down).	Services	No explicit definition of granularity	Distinction between process and basic services, but no detailed specifications	Functional	No	Real world case study
Fiege 2009	Modeling a service-oriented architecture by using Axiomatic Design following the architectural goals of loose coupling, high autonomy, and balanced granularity.	Axiomatic Design: structured top-down method based on business processes.	Services	Granularity describes the scope and type of the functions. The functional complexity is measured by the sum of the data flows of the service operations.	Decomposition rules and modeling guidelines	Functional	No	Three real world case studies
Her et al. 2008	Identification of services based on use cases.	Five-stage procedure for the deduction of service specifications in use cases and business processes.	Services	No explicit definition of granularity	Granularity is already defined for use cases. If these are included in other applications, a sub-process and then a so-called composite service is identified.	Functional	No	Case study
Kim and Chang 2004	Identification of components with regard to cohesion and coupling.	Clustering method on the basis of use case diagrams (bottom up).	Components	Number of realized use cases (implicit)	Granularity depends on determining the parameter values of the clustering method (no optimization).	Functional, partly technical	No	None (only discussion and evaluation in comparison to other approaches)
Lee et al. 2001	Identification of components with regard to cohesion and coupling.	Clustering method on the basis of use case diagrams (bottom up).	Components	Number of classes (implicit)	Granularity depends on the cohesion and coupling of the classes and on determining the parameter values of the clustering method.	Functional and technical	No, only clustering-algorithm	Real world case study

Offermann 2008	Design of software artifacts based on a service-oriented architecture.	Top-down analysis consolidated with bottom-up analysis of existing systems.	Services	No explicit definition of granularity	No guidelines	Functional	Yes (but only for service modeling, not for granularity optimization)	Real world case study and laboratory experiment
Wang et al. 2006	Identification of components with a focus on the stability of the business model.	Analysis based on the so-called "Feature Tree", which illustrates the features (of a domain) and their dependencies and differentiates them by their stability.	Components	Number of features / functions or implemented entities	Optimization of the granularity based on the "Feature Tree"	Functional and economic (cf. also Wang et al. 2005)	Declared as "future work"	Qualitative comparison with other identification approaches and illustration of this comparison by a simple example
Winkler 2007	Identification of services with a focus on reuse.	Functional analysis by gradual decomposition of activities in activity diagrams (top down).	Services	Orientation on the levels of decomposition (implicit)	Decomposition rules	Functional	No	Real world case study
Winter 2003; Schelp and Winter 2008	Identification of services which belong together from a functional point of view.	Analysis of relationships between data objects and functions and their clustering based on a matrix (multidimensional).	Services	Functionality of a service (implicit)	Decomposition and clustering rules	Functional	No	Real world case studies with four enterprises

Approaches dealing with component identification

In their *Business Component Identification* approach Albani et al. (2008) identify (business) components based on information objects, process models and the relationships between them. By means of a clustering method, a partitioning of components is performed. In the approach preferences are considered as well as the type and frequency of relationships between information objects and actions of the process model. Thereby, the design principles of high cohesion and low coupling are taken into account.

The granularity question also constitutes an objective explicitly stated by other clustering approaches (cf. Kim and Chang 2004; Lee et al. 2001). Those are based on mathematical-statistical methods which measure the cohesion or coupling of a system's discrete elements – often classes in terms of object orientation. For instance, the number of reciprocal relationships between these elements is derived and from that loosely coupled clusters are deduced which hold a high cohesion each. Subsequently, these clusters constitute the searched components. Depending on the determined parameter values of the clustering method, components of different granularity are identified. However, the choice of parameter values, particularly in economic terms, is not explicitly addressed in the contributions mentioned before.

Wang et al. (2006) develop their STCIM (Stability Based Component Identification Method) approach with the objective to primarily identify the components with regard to different degrees of stability of (parts of) the business model. Stability as a parameter is defined as the extent and the number of business-related changes, i.e. the fewer changes there are, the more stable are the parts of the business model and the less must the associated components be adjusted. Stability is therefore seen as an indicator for the design of coarse-grained components and vice versa (e.g., Wang et al. 2006, p. 2). To achieve these objectives, Wang et al. (2006) define a tree structure (so-called "Feature Tree") which reflects the result of a domain analysis. This tree structure contains the features (in terms of functions) and their relationships. This means that features are gradually refined in order to define the different levels of the tree. Depending on what level of the tree a component is implemented at, components of different granularity result.

In addition, the economic terms of composition resp. change costs, which are explicitly discussed by Wang et al. (2006, p. 6), are of particular interest. It is suggested that the more coarse-grained components are, the less is the effort of the composition and vice versa. In contrast, the change costs increase the more coarse-grained components are and vice versa. Both statements, however, are based on discussions; the exact functional relationship to calculate, e.g., the composition costs, is not defined or executed.

Moreover, Wang et al. (2005, p. 231) present an optimization calculus for the identification and design of components. Here, different objectives and their optimization rule (minimizing or maximizing) are defined for a component *c*, such as Reusability (variable *R(c)*), Reuse Costs (*RC(c)*) or Reuse efficiency (*RE(c)*). These objectives are then integrated into a single objective function which has to be maximized for the set of all components *c*. Unfortunately, no functional relationship for calculating the individual objectives are defined in Wang et al. (2005, p. 231). In this respect it is hardly comprehensible what the properties and elements of the economic calculus are or how it is to be used for accurate component identification. In addition, it should be noted that the individual terms have different measurement units. For example, the Reuse Costs (*RC(c)*) should be measured in monetary units, whereas variables, such as *Cohesion(c)* or *Coupling(c)*, are defined as "semantic proximity" (without specifying the measurement unit). Hence, it is unclear how the combination of the individual terms in the presented objective function leads to an interpretable overall result.

Approaches dealing with service identification

As opposed to some approaches on component identification, **Table 1** shows that service identification focus almost exclusively on a domain analysis while precise rules for defining the granularity of services are specified only partly.

For example, Aier (2006) suggests a clustering algorithm for a modularization of an application system landscape from a functional view. Among other things, it is supposed to determine service granularity in this way. Winter (2003) draws upon IBM’s Business Systems Planning approach (1984) and proposes three dimensions for the structuring of an application system landscape (function, information object, performance/organization), which has been also transferred to service identification in subsequent work (cf. Schelp and Winter 2008). In IBM’s SOMA (Service-Oriented Modeling Architecture) approach (Arsanjani et al. 2008) as well as in the work by Offermann (2008) services are identified and designed in both ways: bottom up – starting with existing applications or components – and in a top down manner, for example, starting from business process models or business objectives. Winkler (2007) splits activity diagrams in several iterative steps into atomic basis functions or actions in order to decide which of them should be implemented individually or integrated into a service. This procedure is similar to the approach of Her et al. (2008) who also identify services based on process models and use cases through iterative refinement. Beverungen et al. (2008) additionally consider whether a process step is supposed to be visible for business partners to identify services. Fiege (2009) conveys the so-called Axiomatic Design, a method originating from industrial production, to service identification. Possible solutions for fulfilling the pre-defined requirements are represented in the form of matrices which are refined in a top down approach by assignment and decomposition. The indicated relationships between functional requirements and design parameters are supposed to enable the identification and design of services under the premise of loose coupling, high autonomy, and "balanced"

granularity.
The application of many approaches mentioned above offers degrees of freedom regarding service identification. Depending on the extent to which, for example, activity diagrams are refined or features are disaggregated, services of varying granularity can emerge. At this point, the present work starts on the question of how these available degrees of freedom in terms of service granularity can be used under economic aspects. Insofar the above mentioned work will be extended. This corresponds to a two-stage approach:

1. Applying a functionally-oriented approach for service identification leads to alternative service candidates (representing degrees of freedom) which differ in service granularity.
2. An economic optimization – which takes into account development and maintenance costs as well as reuse potential – has to be performed to leverage the available degrees of freedom and finally determine the optimal service granularity.

Some approaches for modeling SOA already include an explicit step to consolidate alternative service candidates (e.g., Offermann, 2008, p. 467) which can be extended to include an economic assessment. None of the previously investigated approaches, however, offers such an economic optimization. This is also valid for the component domain. Although in particular the work of Wang et al. (2005) presents already some economic calculus, these discussions are hardly suited for answering the question of service granularity since objective functions and functional relationships (e.g., to what extent do the costs of service implementation depend on the size of functions?) are not defined or substantiated. In the following we therefore present an approach taking these issues into account.

3 Granularity of services – An economic analysis

The definition of a Functionality and Service Graph (FSG) in section 3.1 represents the starting point for the presentation of granularity metrics (section 3.2) and the economic analysis in section 3.3.

3.1 Functionality and Service Graph (FSG)

Some of the approaches presented above propose to identify services based on an aggregation or disaggregation of functions or functionality. This raises the question of how the results of such an analysis can be represented in a suitable way for our purpose. Below, a graph – the so-called FSG – is defined for the representation of these results. The FSG is supposed to represent the disaggregation relationships between functions. Compared to, for example, Wang et al. (2006) some enhancements are required for the identification of services:

1. To represent a multiple use of functionalities, we use a directed acyclic graph instead of a

tree. In Wang et al. (2006, p. 4), the "feature tree" has the typical characteristics of a tree structure, i.e. a son functionality is restricted to have only one parent functionality.

2. Services can be composed to perform various processes. This requires that the graph is able to contain more than one root (i.e. a source node with no incoming edges) which are to be interpreted as processes. In Wang et al. (2006, p. 4) the "feature tree" is restricted to have only one root, i.e. several processes can not be represented as different source nodes.

The assumptions and definitions for the FSG are as follows:

(A1) The FSG is a directed acyclic graph $G=(N, E)$. The functionalities $m \in M$ *form a subset of the set of nodes* N $(M \subseteq N)$. *The* disaggregation relationship between two functionalities m_i and m_j is represented as a directed edge $(m_i, m_j) \in E$. The disaggregation of a functionality m_i into the functionalities m_j, …, m_{j+n} is defined as disjointly and completely.

A directed edge (m_i, m_j) implies that "functionality m_j is part of functionality m_i". In the FSG all functionalities and disaggregation relationships of the considered domain are represented.

Every source node of the FSG, i.e. a node without incoming edges, is called a *process* (with P as the set of all processes, $P \subseteq M$). Every sink node of the FSG, i.e. a node without any outgoing edges, is called a *basic functionality* (with B as the set of all basic functionalities, $B \subseteq M$). The inner nodes of a graph, i.e. functionalities, which are neither processes nor basic functionalities, are referred to as *preceding functionalities*. $V \subseteq M$ is the set of all preceding functionalities.

In addition, a sequence of nodes and edges $m_0, (m_0, m_1), \ldots, (m_{n-1}, m_n), m_n$ is referred to as a path $w(m_0, m_n)$ with the starting node m_0 and the end node m_n. If the starting node of a path is a process (i.e. $m_0 \in P$) and if the end node is a basic functionality (i.e. $m_n \in B$), this corresponds to a *complete path*. The distance $d(m_0, m_n)$ of a path $w(m_0, m_n)$ is defined as the number of edges of the path w in the acyclic FSG.

The disaggregation relationships between functionalities and therefore the paths can be depicted in form of an adjacency matrix I_{MM}: $M \times M$ whereas $I_{m_i,m_j} = 1$ is valid if $(m_i, m_j) \in E$ exists. Otherwise, $I_{m_i,m_j} = 0$ holds.

The result of our economic decision model is supposed to be shown in the FSG as well, i.e. what functionalities concretely have to be realized by a service:

(A2) Every service $s_i \in S$ with $S \subseteq N$ is allocated to exactly one functionality $m_j \in V \cup B$ through a directed edge $(s_i, m_j) \in E$. A service s_i fully implements the functionality m_j, including all functionalities m_k for which there is a path $w(m_j, m_k)$ and exposes exactly this functionality via its interface.

A functionality m_j for which $\exists (s_i, m_j) \in E$ applies is called *implemented functionality*. A multiple implementation of a functionality occurs if a functionality is implemented by various services either directly or indirectly (i.e. through preceding functionalities). The matrix I_{SM}: $S \times M$ represents the allocation of services, where $I_{s_i,m_j} = 1$ exactly applies if $(s_i, m_j) \in E$ with $s_i \in S$ and $m_j \in V \cup B$ holds. Otherwise, $I_{s_i,m_j} = 0$ applies.

The above stated definitions are illustrated in **Fig. 1** using a simple example. As a result of a domain analysis the disaggregation of two processes m_0 and m_{10} into the functionalities m_1 and m_4 resp. m_{11} etc. is depicted. The functionality m_9 is required in both processes. A *possible* implementation of the functionalities by means of the services s_1 to s_5 is also shown. For instance, the service s_3 directly implements the functionality m_6 and thus also realizes the basic functionalities m_8 and m_9. Since the service s_5 also implements the functionality m_9 through the functionality m_{11}, m_9 is implemented twice in the example.

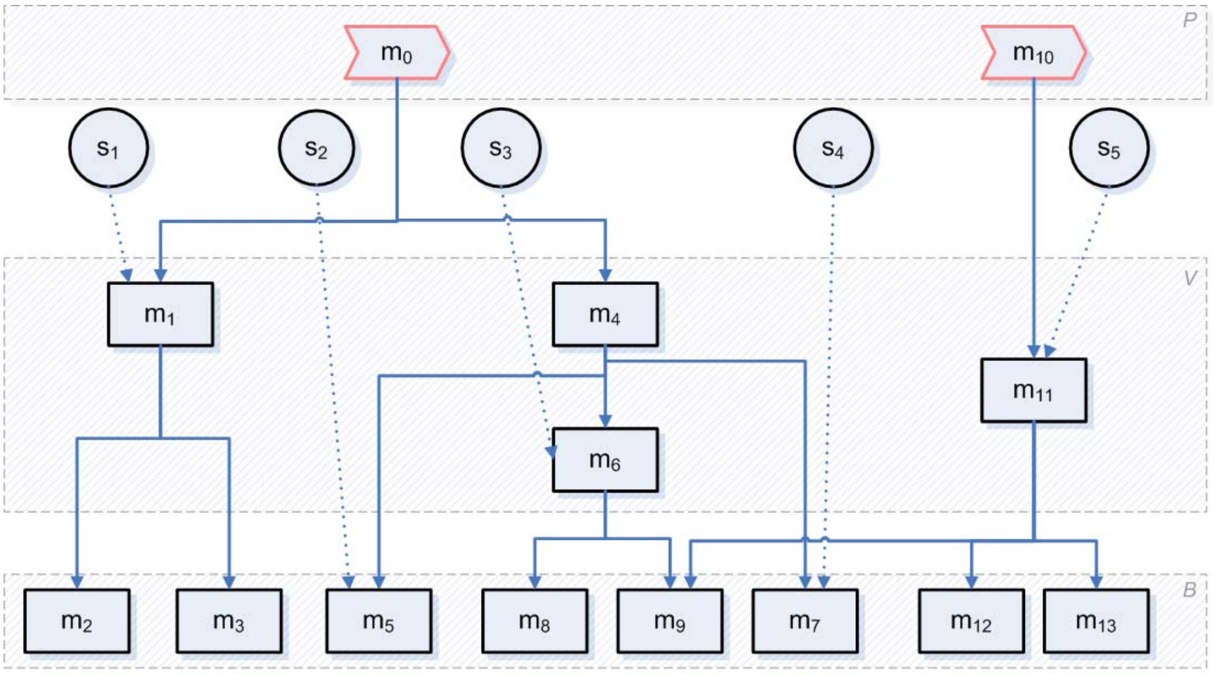


Fig. 1 An exemplary Functionality and Service Graph (FSG)

3.2 Granularity metrics

Existing approaches oftentimes make statements about granularity – such as coarse- vs. fine-grained services – based on an implicit understanding of granularity; which means only few authors define this term explicitly or mathematically (see **Table 1**). In the following, we therefore propose three metrics of granularity to operationalize different perspectives and discuss their advantages and disadvantages. With the help of such metrics the granularity of different services can be measured and compared. For the metrics calculation we use the definitions of the FSG. The starting point is a service s_i which implements a functionality m_j and the granularity of this service s_i has to be determined.

Distance-oriented metric

First, a distance-oriented metric is presented. Basically, it indicates the position of the service s_i and thus of the implemented functionality m_j within the FSG. However, a simple calculation of the path distance starting from the process to the realized functionality is not sufficient. This has two reasons: First, this value is not very meaningful. For example, the same granularity for two services might result from the calculation of this value although one of the services implements a basic functionality, while the other service implements a preceding functionality with many subsequent functionalities. Second, the determination of the path distance based on a graph – instead of a tree – is not clear if a functionality is included in several paths. Both issues motivate the development of the following metric for operationalizing the distance-oriented metric: The metric is based on the path distance from the process to the implemented functionality in relation to the distance of the complete path. For an implemented functionality m_j, which is part of the complete path $w(m_p, m_n)$, *the* metric is calculated as follows: $z_w = \frac{d(m_p, m_j) - 1}{\max[1, d(m_p, m_n) - 1]}$. In the numerator as well as in the denominator we have to subtract one, since the path distance is calculated in each case starting from the process $m_p \in P$. The range of values of z_w is normalized to the interval *[0; 1]*. Thus, the implementation of a basic functionality (maximum fine-grained) leads to the value one, while the value zero results from the realization of a functionality that follows directly after a process m_p (maximum coarse-grained). The metric value z_w is calculated for all paths which include the implemented functionality m_j. For those values*,* the arithmetic mean is taken. Let *W* be the set of all paths *w*, which contain m_j, then the following applies for the depth metric: $g_T(s_i) = \frac{\sum_{w \in W} z_w}{|W|}$. The metric is normalized to the range *[0; 1]*.

Scope-oriented metric

The distance-oriented metric has disadvantages, for example, if an implemented functionality has many directly and indirectly following functionalities. In this context, the scope-oriented metric returns meaningful values. Here, a service is the more coarse-grained, the more functionality it implements in total (direct and indirect).

This metric is operationalized by means of the number n_{m_j} of directly or indirectly implemented functionalities through a service. This value is divided by n_{m_a}, with m_a as the functionality directly following the process, and subtracted by one: $z_w = 1 - \frac{n_{m_j} - 1}{\max(1; n_{m_a} - 1)}$. Analogously to the distance-oriented metric, the value range is normalized to *[0, 1]*. If a basic functionality is implemented by a service, a value of one results (because $n_{m_j} - 1 = 0$). In the case of implementing a functionality directly following the process, the values n_{m_j} and n_{m_a} correspond to each other. Hence, it follows that z_w equals zero (except for a basic functionality). The normalization to the interval [0, 1] constitutes an advantage compared to the metrics of Wang et al. (2006, p. 5-6) and to Haesen et al. (2008) and their "default functionality granularity". In both approaches a metric is proposed without normalization which limits the comparison of metric values for different services.

If a functionality m_j is part of several paths starting from *different* source nodes m_p or preceding functionalities m_a, the arithmetic mean g_{BT} can – similar to above – be calculated based on the values of these paths.

Although both metrics discussed so far are already more meaningful – compared for instance to measurements by Wang et al. (2006) – they rely purely on the number of functionalities. In other words, even if two services show an identical granularity for both metrics, they can still differ significantly regarding the size of the implemented functionalities. This leads to a third metric.

Size-oriented metric

The question of how the size of a functionality can be measured or estimated ex ante before its implementation has already been studied in the literature about effort estimation. Here, measurement units, such as lines of code (LOC) or Function Points, are used. LOC are used in the COCOMO approach, which is a method for effort estimation of software development projects, and indicate how many lines of source code for a program may need to be written (cf. Boehm et al. 2000; Wehrmann and Gull 2006). In COCOMO, the calculation of the person-months is founded on the LOC of the software artifact together with the costs disseminators, the scaling factors as well as the individual calibration factors. To determine the LOC either historical data from other projects, experts estimates, or algorithmic procedures can be used. Cost factors are then included in the calculation by multiplication whereas scale factors are considered exponentially. In the case study illustrated below, LOC were chosen, while other units can be used in both metric and in the decision model as well.

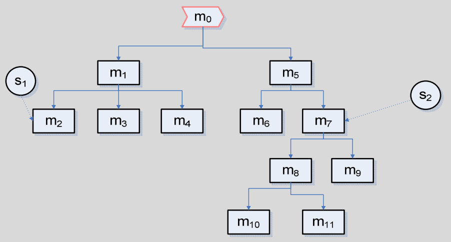
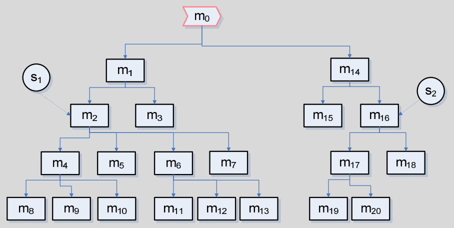
A prerequisite for this metric is the estimation of the $size_{m_j}^{func}$ (e.g., in LOC) for a basic functionality m_j. The size of the preceding functionalities then results from the size of the subsequent functionalities (disjoint and complete disaggregation according to (A1)) plus the $size_{m_j}^{comp}$ of the composition logic (cf. also Erl 2007). The latter contains information for the control, integration, and the subsequent invocation of functionalities. Hence, the $size_{m_j}$ of the functionality m_j is:

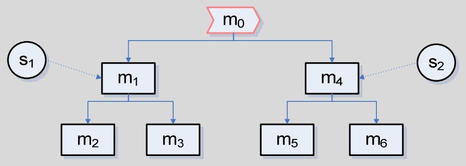
$$size_{m_j} = \begin{cases} size^{func}_{m_j} & \text{if } m_j \in B \\ size^{comp}_{m_j} + \sum_{i=1}^{|M|} I_{m_j,m_i} \cdot size_{m_i} & \text{if } m_j \in V \end{cases}$$

For an implemented functionality m_j, which is part of the complete path $w(m_p, m_n)$ with m_p, (m_p, m_a),…, (m_{n-1}, m_n), m_n, the size-oriented metric is defined as $z_w = 1 - \left[\frac{size_{m_j}}{size_{m_a}}\right]$. If $m_j = m_a$ holds, which means a functionality m_a directly following the process m_p is implemented, then z_W is zero indicating a maximum coarse-grained service implementation. The value z_W becomes larger when implementing a basic functionality. This provides a range of $[0, 1[$ for z_W. If a functionality m_j is part of several paths, again, the arithmetic mean g_G can be calculated by the values of those paths.

In addition to the first two metrics the size-oriented metric can provide an additional value especially when the difference between the sizes of the implemented services differs strongly. **Table 2** summarizes the metrics.

Table 2 Granularity metrics

Metric	Description and Definition	Applicability / Restrictions for the application	Illustration of the idea of the metric (examples)
Distance-oriented metric	The metric measures service granularity considering the position of the implemented functionality in the FSG: distance of the path from a process to the implemented functionality in relation to the distance of the complete path.	1. The metric refers to the paths of the FSG and leads to results which are easy to interpret if two or more implemented functionalities contain (almost) the same number of directly and indirectly following functionalities. 2. The metric value may be hardly meaningful if the implemented functionalities differ strongly regarding their size (e.g., LOC).	 m0 s1 m1 m5 s2 m2 m3 m4 m6 m7 m8 m9 m10 m11 Due to the different distance of the complete paths, the service s_2 with $g_T(s_2)=0.39$ is more coarse-grained than the service s_1 with $g_T(s_1)=1$ according to the distance-oriented metric. In contrast, a mere comparison of the number of preceding functionalities would show the same granularity in this example.
Scope-oriented metric	The metric measures service granularity by the number of directly and indirectly following functionalities.	1. The metric leads to results which are easy to interpret if service implementations are compared which differ in terms of their number of directly and indirectly following functionalities. 2. The metric value may be hardly meaningful if the implemented functionalities differ strongly regarding their size (e.g., LOC).	 m0 s1 m1 m14 s2 m2 m3 m15 m16 m4 m5 m6 m7 m17 m18 m8 m9 m10 m11 m12 m13 m19 m20 In this example, both services s_1 and s_2 show the same granularity value according to the distance-oriented metric (g_T). Considering the scope-oriented metric, however, there are comprehensible differences with $g_{BT}(s_1)=1/6$ vs. $g_{BT}(s_2)=1/3$. That means that s_1 is significantly more coarse-grained compared to s_2.

Size-oriented metric	The metric measures the granularity of a service by its size (e.g., measured in LOC)	1. The metric indicates the differences in the size of the implemented functionalities in the FSG. 2. The metric value is meaningful if the granularity is used as an indicator for the implementation costs of a service – as expressed for example in LOC.	 m_0 s_1 m_1 m_4 s_2 m_2 m_3 m_5 m_6 According to both metrics above, the services s_1 and s_2 are equally granular. If however the implemented functionalities m_1 resp. m_4 are different in terms of their size, this becomes evident only through the size-oriented metric.

The presented metrics systematize different (often implicit) understandings of granularity in the literature. For instance, Winkler (2007) evaluates the decomposition relationships in the course of her proposed procedure in a way similar to the distance-oriented metric. The understanding conceptualized by the scope-oriented metric can be similarly found, e.g., in Aier (2006) and Fiege (2009). The latter explicitly discusses the number of implemented functionalities (scope-oriented metric) in addition to the abstraction level of services (similar to the distance-oriented metric). Clustering methods that aggregate elements such as classes to components suggest an implementation-related granularity definition similar to the size-oriented metric. However, this is not made explicit in most cases.

Based on these metrics we can not only measure the granularity of individual services. If we aggregate the granularity value of all services throughout the entire FSG, we may specify a metric value for entire service landscapes (see also the software tool presented in section 4). In this way, different solutions resulting from applying the economic decision model can be analyzed or compared regarding their service granularity value. That means that the metric values are *not* used as input to apply the decision model. Instead, the resulting model output is assessed by means of the metrics in terms of its granularity (for example, to evaluate whether a solution is fine- or coarse-grained).

3.3 Economic Decision Model

In order to contribute to the research question of how granular services should be defined in economic terms, it is important to identify the relevant cost factors. We deliberately limit ourselves to the costs since the functionalities (which are supposed to lead to profits) are determined after the completion of the domain analysis and are represented in the FSG. Based on this, we suggest to leverage the available degrees of freedom of the domain analysis for a cost optimization.

Here, on the one hand the *costs of the implementation of a service* are relevant for decision-making. Usually the implementation costs increase at a higher rate than the size of a service due to the resulting increased complexity of the implementation (for instance side effects are more complex to be handled for larger services). Also, testing is more cost-intensive with an increasing size of services. On the other hand, the *costs of service composition* using languages such as WS-BPEL have to be included in the decision. Here, for example, the costs for searching and integrating individual services, the creation costs for a WS-BPEL file, or the preparation of a subsequent operation of the composition are included. Finally, it is important to take *maintenance costs* into account which are influenced by the choice of service granularity. For instance, if a functionality is implemented redundantly in several services multiple maintenance costs occur.

On the contrary, *one-time costs* for establishing a service-oriented infrastructure are not considered relevant to our decision problem because the corresponding effort is independent of the choice of service granularity: This aspect includes the introduction of service standards, setting up and installing the infrastructure (e.g., engines, directory servers, enterprise service bus), the installation and introduction of the development environment, etc.

Objective function

Our decision problem can be described as follows: A feasible solution (the criterion of feasibility is described below) for the allocation of services to functionalities (as represented by the matrix I_{SM}) is supposed to be found which minimizes the total costs of the implementation C_R, the composition C_K, and for maintenance and support C_P:

$$Z(I_{SM}) = C_R(I_{SM}) + C_K(I_{SM}) + C_P\left(I_{SM}\right) \rightarrow \text{min!}$$

Each cost factor C_R, C_K, and C_P is described below.

Costs of service implementation

The starting point for estimating the implementation costs of a service is – as discussed above – the size of the functionality to be implemented. The following assumption is made:

(A3) For realizing a service s_i, costs $c_R(s_i)$ arise which depend on the $size_{m_j}$ of the functionality to be implemented – e.g. quantified in LOC. We assume a cost rate c_{var} per LOC. Furthermore costs increase more than proportionally (exponent $b>1$) to the size (for reasons of complexity). In addition, costs c_{fix} which are independent from the size may occur for the service implementation (e.g., deployment costs, such as publication of a service in the service directory).

The parameters c_{var} and b may, for example, be interpreted as linear and scaling factors similar to the COCOMO approach. Based on (A3), the implementation costs of a service s_i yield to:

$$c_R(s_i) = \sum_{j=1}^{|M|} I_{s_i,m_j} \cdot (c_{fix} + c_{\text{var}} \cdot (size_{m_j})^b) \quad \text{with}: c_{\text{var}} > 0,\ c_{fix} \geq 0, b > 1$$

Moreover, it should be pointed out that for different service types the values of the parameters c_{var} and b can be varied. However, for reasons of a clear presentation, we abstained from an additional indexation of these parameters that would be necessary in this case. The prototypical software presented below allows for such an enhancement.

Thus, the total implementation costs of a service landscape results from summarizing the implementation costs of all services:

$$C_R(I_{SM}) = \sum_{i=1}^{|S|} c_R(s_i)$$

Costs of service composition

If a process is implemented through many services instead of a few services, the composition costs increase. Here, we use the previously defined size of compositional logic as the starting point for our analysis:

(A4) The costs for service composition c_K of a process depends on the size of the composition logic which has not been directly implemented through a service yet (at a cost rate of c_{var}^{comp}). It is assumed that the costs increase more than proportionally (exponent $f>1$) to the size of the composition logic (due to a higher complexity).

Assumption (A4) can easily be illustrated with the help of **Fig. 1**: For the implementation of the process m_0 the services s_1, s_2, s_3 and s_4 have to be composed. In addition, the functionality m_4 and therefore $size_{m_4}^{comp}$ has to be considered for the composition logic (since the composition logic of m_4 is not implemented by a service) as well as $size_{m_0}^{comp}$ for the process m_0 itself. Thus, the composition costs required for a process $m_p \in P$ generally result in:

$$c_K(m_p) = c_{\text{var}}^{comp} \cdot \left(size_{m_p}^{comp} + comp(I_{SM}, m_p)\right)^f \quad \text{mit}: c_{\text{var}}^{comp} > 0, \quad f > 1$$

Here, the values of the parameters c_{var}^{comp} and f may differ from the parameter values of c_{var} and b due to different methods and languages used for the implementation of services and for their composition respectively. With the help of $comp(I_{SM}, m_p)$, the size of the preceding functionalities that have not already been implemented by a service (such as m_4 in the example) are determined through the paths from m_p to the basic functionalities. The entire composition costs are obtained by summarizing the composition costs over all processes.

Maintenance and support costs for multiple implementations

In addition, those maintenance and support costs are relevant for our decision problem which can be avoided explicitly through the choice of service granularity. If a functionality needed in several processes were implemented by different services, the functionality of each service implementation has to be adapted with each necessary change. Even if this is contrary to the basic idea of SOA, neglecting this case would be unrealistic. In this respect, it is proposed to make those costs of maintenance and support to depend on the number and size of *multiple* implemented functionalities. The reason for this is that with an increasing number and an increasing size of a multiple implemented functionality, maintenance and support costs are expected to increase, too, because of the rising complexity (cf. Keller 2007):

(A5) Any multiple, redundant implementation of a functionality creates additional costs of maintenance and support which increase more than proportionally compared to the size of functionality. Similar to above, we propose a value $pen_{var} > 0$ for the variable costs and an exponent $h>1$.

Based on (A5), we determine how often a functionality m_i is implemented directly and indirectly by different services. In **Fig. 1**, the number of implementations of the functionality m_9 has to be determined with $r_{m_9} = 2$. The additional maintenance and support costs c_p for the functionality m_i result in:

$$c_p(m_i) = \max[(r_{m_i} - 1), 0] * pen_{\text{var}} \cdot (size_{m_i})^h \quad with : pen_{\text{var}} > 0, h > 1$$

The *max*-function assures that only multiple implemented functionalities are included. The entire additional maintenance and support costs C_P resulting from a multiple implementation is calculated again by summarizing the costs $c_p(m_i)$ over all functionalities.

Choice of service granularity from an economic perspective

Using the FSG and the above introduced objective function the service granularity can now be optimized under economic aspects. Here, the basic functional relationships are considered: the more fine-grained a service is (e.g., according to the size-oriented metric), the more probable it is that it can be re-used without having to realize the functionalities implemented by this service multiple times. This reduces ceteris paribus the implementation costs. **Fig. 1** shows an example: The functionality m_9 is needed for two processes, but it is not directly implemented. Hence it is necessary to consider the size of this functionality for the calculation of the implementation costs of two implemented functionalities (here m_6 and m_{11}). Thus, the implementation costs arise twice for the services s_3 and s_5, whereas the implementation costs for the functionality m_9 would only arise once in case of a direct implementation. In addition, multiple implementations lead also to increased maintenance and support costs. However, higher costs of composition occur in case of fine-grained services.

The determination of the optimal solution is not trivial. The adjacency matrix I_{SM} has to be specified in the way that the solution of the decision problem is feasible and minimizes the total costs according to the objective function. To identify the solution with the minimal total costs, an algorithm is implemented which gradually allocates services to the functionalities

starting with the basic functionalities. Subsequently, the services that implement basic functionalities are substituted iteratively by services that implement preceding functionalities. For each of the solutions we have to check its feasibility. An allocation of services to functionalities in the FSG is feasible if it is possible to compose and execute all processes with the allocated services. This means specifically that each complete path $w(m_0, m_n)$ in the FSG must contain at least one directly implemented functionality. This corresponds to the feasibility criterion and is sufficient because a preceding functionality always directly or indirectly contains the basic functionalities referenced by the disaggregation relationships (adjacency matrix I_{MM}). If a solution is not feasible, there is at least one basic functionality $m_n \in B$ of a complete path that is not employable for the process $m_p \in P$. Process m_p would therefore not be executable. These conditions for a feasible solution have been defined mathematically (see Appendix 1) and realized in our prototypical software tool. Finally, the solution whose realization leads to the minimal total costs is selected out of all feasible solutions.
For the calculation also the input parameters of the objective function have to be determined. Here, we can draw upon known estimation procedures from the field of software development. In the COCOMO approach, costs are estimated by using the formula $PM = A \cdot EM \cdot size^{B+SF}$, with A and EM being linear factors and B and SF being scaling factors. Their values are to be determined project- and company-specific. COCOMO has already been used for service implementations by Tansey and Stroulia (2007). They illustrate the general applicability but also point to the fact that it is often not possible to use large existing databases to estimate the linear and scaling factors. Here one has to – as the following case study will also show – rely on company-internal estimates. Nevertheless, basic assumptions, such as the convex slope of the cost functions, appear reasonable. To analyze the effects of imprecise estimates, a sensitivity analysis has been integrated in our prototypical software tool which supports the investigation of the robustness of the results.

4 Prototypical implementation and case study

In the following we illustrate the implementation of the decision model in a software tool. Afterwards, the case study of a financial service provider is discussed.

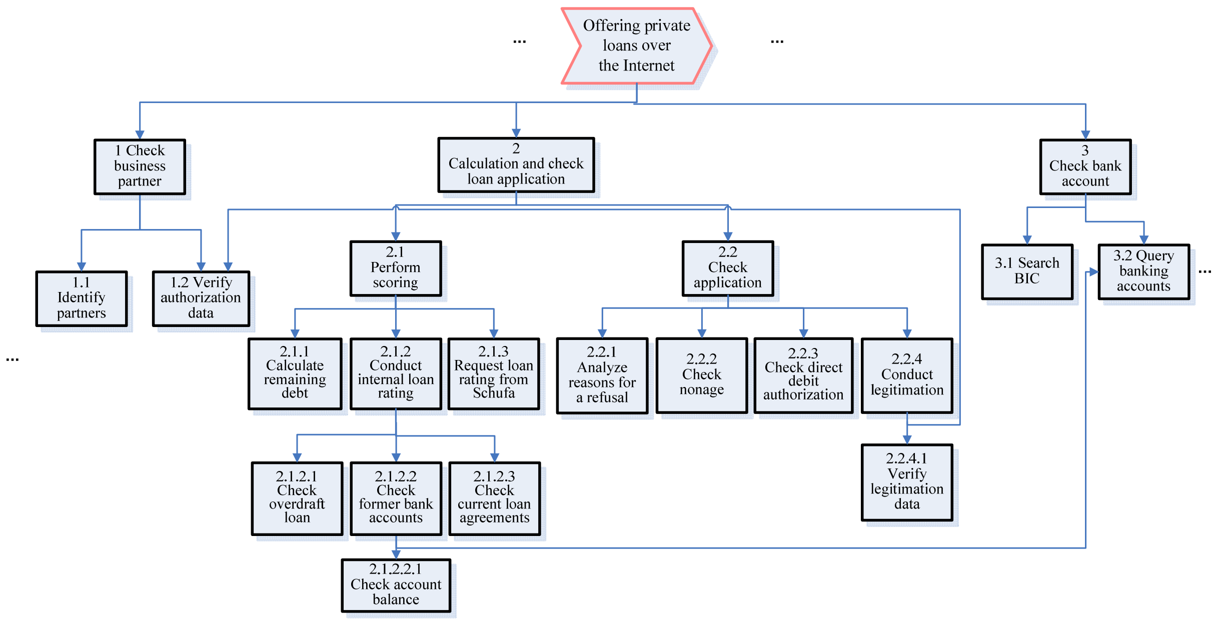
4.1 Prototypical implementation

Our model was implemented as a plug-in for the open-source framework Eclipse. The implementation was carried out in Java and is based on the Eclipse Modeling Framework. The graphical input and output were realized using the Graphical Editing Framework. For creating the FSG, a graphical editor is available in which the functionalities are drawn from a side menu (see right part of **Fig. 3**) into the working area. The functionalities can be integrated in the acyclic FSG using directed edges. Another screen allows to enter parameter values of

the cost functions and allows these functions to be changed individually for each company (e.g., to use function points instead of LOC).
In the next step, the service implementation with minimal costs is determined based on this data. The results are twofold: On the one hand, results are illustrated in the form of a table that shows which functionalities should be realized as a service, the associated total costs of the solution, and the values of the three granularity metrics. On the other hand, the minimal cost solution is also illustrated graphically (see **Fig. 3**). Here, each functionality that has to be implemented according to the determined solution is connected to a service (symbolized by a yellow circle). This kind of presentation is especially useful for a quick comparison of alternative solutions. As discussed, the estimate of the parameter values can be imprecise. Therefore, the software tool also provides a sensitivity analysis. For each parameter input fields are available in order to specify an interval, which means a lower and upper limit. With these inputs we can analyze whether a determined solution with minimal total costs changes when taking these intervals into account. Using the sensitivity analysis it is also possible to automatically conduct a gradually change of a single parameter value until a new optimal solution is found. This also gives insights into the robustness of identified solutions, which turned out to be of great value in the practical application.

4.2 Case study

The standardization of processes and IT applications is currently promoted for financial service providers with the aim of reducing costs, among other. Here the replacement of monolithic legacy systems through service-oriented application systems is also increasing in importance (for more SOA objectives see Baskerville et al. 2010). Against this background, a SOA has been introduced in the loan division of a major German financial service provider. In a first step, a domain analysis has been carried out, which means functionalities have been identified based on the processes that had to be implemented. However, this analysis left open significant degrees of freedom in terms of alternative service candidates. In order to avoid deciding only intuitively or according to rules of thumb, economic aspects had to be included. As an example, a part of a loan process is considered (an excerpt from the loan approval process “*Offering private loans over the Internet*"), which is illustrated in **Fig. 2** in a simplified form and anonymized for confidentiality reasons.



Offering private loans over the Internet
...
...
1 Check business partner
2 Calculation and check loan application
3 Check bank account
1.1 Identify partners
1.2 Verify authorization data
2.1 Perform scoring
2.2 Check application
3.1 Search BIC
3.2 Query banking accounts
...
2.1.1 Calculate remaining debt
2.1.2 Conduct internal loan rating
2.1.3 Request loan rating from Schufa
2.2.1 Analyze reasons for a refusal
2.2.2 Check nonage
2.2.3 Check direct debit authorization
2.2.4 Conduct legitimation
...
2.1.2.1 Check overdraft loan
2.1.2.2 Check former bank accounts
2.1.2.3 Check current loan agreements
2.2.4.1 Verify legitimation data
2.1.2.2.1 Check account balance

Fig. 2 Part of the FSG in the case study

The process “*Offering private loans over the Internet*“ is disaggregated into three functionalities: “*Check business partner*” includes the functionalities for the identification and verification of a partner. Below "*Check bank account*" the functionalities “*Search Bank Identification Code (BIC)*” and “*Query banking account*” are located. The third functionality "*Calculation and check loan application*" is disaggregated into "*Perform scoring*" and "*Check application*". As shown in **Fig. 2**, some functionalities such as "*Verification authorization data*" and "*Verification legitimation data*" are used twice.

In order to determine the size of the functionalities measured in LOC, already implemented parts of the source code or existing documentation has been used. The size of non- or inadequately documented functionalities was estimated. This was based on the experience of internal cost estimates of previous projects (e.g., based on the complexity of functionalities, the processed data or the development processes used). Finally the LOC shown in **Table 3** resulted for the above functionalities. Here, the LOC for the preceding functionalities only include the composition logic and do not represent already aggregated values (the functionalities are referenced by using the numbers introduced in **Fig. 2**):

Table 3 Size (LOC) of functionalities in the case study

No. of functionality	*LOC*	No. of functionality	*LOC*	No. of functionality	*LOC*
1	200	2.1.2	600	2.2.2	2,000
1.1	1,500	2.1.3	1,500	2.2.3	800
1.2	1,000	2.1.2.1	1,600	2.2.4	200
2	400	2.1.2.2	200	2.2.4.1	1,700

2.1	1,000	2.1.2.3	1,400	3	400
2.2	450	2.1.2.2.1	400	3.1	1,000
2.1.1	1,400	2.2.1	1,000	3.2	1,400

In addition, the values for the input parameters of the objective function had to be estimated. This was done with reference to the existing COCOMO cost estimation procedure of the financial service provider (in general this can be problematic for several reasons; for the special case of the financial service provider, this appeared quite reasonable). After a thorough discussion, no individual values were defined for the input parameters. Instead, an interval was determined for each input parameter to account for the inherent uncertainty of estimates. **Table 4** shows these intervals:

Table 4 Estimate of the intervals for the input parameters of the objective function in the case study

Input parameters for service implementation	Interval	Input parameters for service composition	Interval
c_{var}	[2.75 - 3.25]	c_{var}^{comp}	[2.75 - 3.25]
b	[1.05 - 1.1]	f	[1.15 - 1.2]
c_{fix}	[80 - 100]		

Background for the determination of the input parameters is the classification of the considered project as a project with an average complexity according to the COCOMO approach (referred to as "semi-detached mode"). This classification was mainly based on an analysis of the experience of the project members with an implementation of SOA and related technologies, the project scope, the quality of the existing requirement specification, the documented interfaces, and the timeframe planned for project implementation. According to COCOMO, a project in semi-detached mode receives a linear factor of 3.0, whereas a project with lower complexity receives 2.4 and a complex project 3.6. The scaling factor of a project in semi-detached mode is 1.12, for a project with lower complexity 1.05, and for a complex project 1.2. The intervals of the cost parameters c_{var} and c_{var}^{comp} (linear factors) were specified with [2.75 to 3.25]. It initially appeared reasonable to set the values for the parameter c_{var}^{comp} higher (compared to the values of the cost parameter c_{var}) since the implementation of the basic functionalities appeared less complex compared to the composition logic. However, the financial services provider decided to choose the same interval for the cost parameters c_{var} and c_{var}^{comp} based on the experience in previous projects. Also in the case of the exponents b and f (scaling factors), the parameter values of COCOMO were used. Here, the intervals were determined with $b \in [1.05 – 1.1]$ and $f \in [1.15 – 1.2]$. By defining these intervals it was possible to analyze whether a service implementation determined as feasible solution is less robust even for small estimation errors. The financial service provider initially refrained from considering additional maintenance and support costs required for multiple implementations.

4.3 Results

Fig. 3 shows the solution with the minimal total costs for the FSG in **Fig. 2**. The services s_1 to s_8 (yellow circles) are assigned to the implemented functionalities (green rectangles).

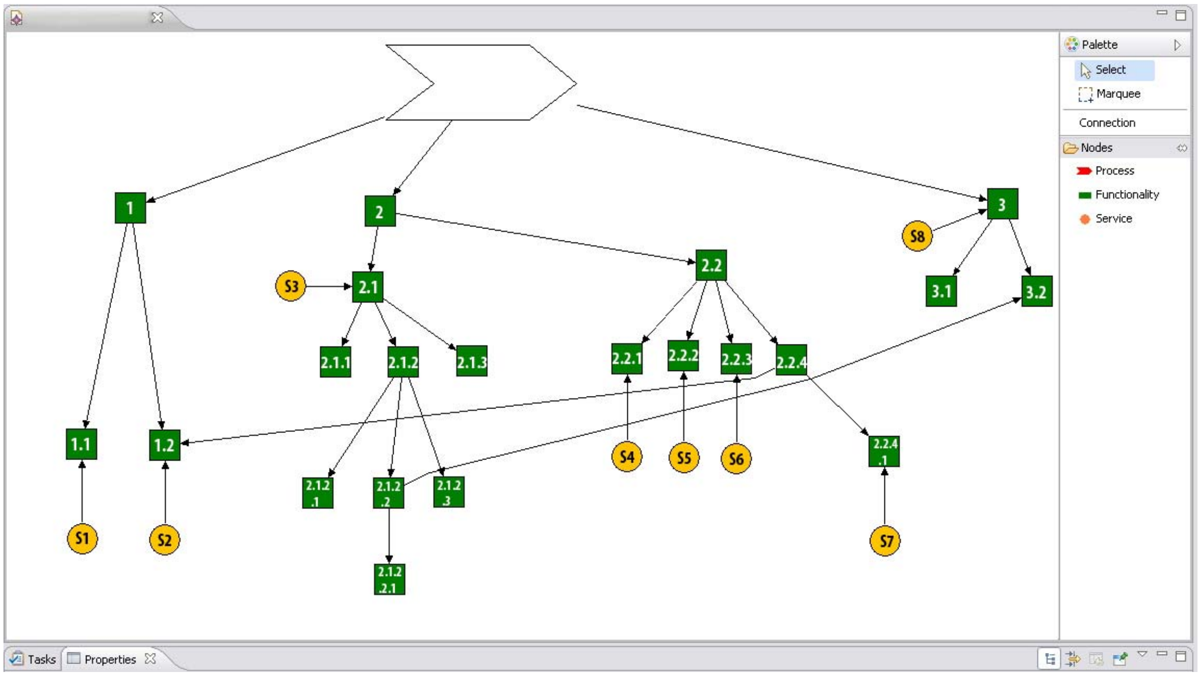


Fig. 3 Assignment of the services in the case study

The above solution, which has total costs of 129,493 MU (monetary units) is relatively fine-grained. The distance-oriented metric shows a value of 0.79, the scope-oriented metric was calculated with a value of 0.81, whereas the size-oriented metric shows a value of 0.67 (for the calculation of these metric values and the total costs see Appendix 2). The functionalities "*Perform scoring*" (2.1) and "*Check bank account*" (3) were implemented with one service each. This leads, for instance, to a multiple implementation of the basic functionality „*Query banking account*". In this example it is easy to see that it is necessary to conduct an individual analysis of the existing FSG. A rule of thumb in terms of a basically coarse- or fine-grained service implementation would not have led to the best economic result since the two "adjacent" functionalities "*Perform scoring*" (2.1) and "*Check application*" (2.2), for example, are implemented fundamentally different. There is one more interesting aspect: If we repeat the optimization several thousand times by means of the software tool, with the values for the input parameters being randomly drawn from the defined intervals, we obtain the following result: The above presented solution remains the minimal cost solution in more than 85% of all runs. In the other 15%, this solution is either the second or third best solution. However, the better solutions differ in a change of one service allocation at the most. Together with the experience-based parameter estimation, this analysis reduces the risk of determining a less robust solution in the case study.

5 Summary and Further Research

In this paper, a decision model was presented that supports the choice of an adequate service granularity from an economic perspective. Such an economic optimization can extend a previous domain analysis. Both, the variety of possible service candidates and the complex cost effects regarding the choice of an adequate service granularity make a manual optimization very difficult or even impossible. Therefore, a software tool has been developed and its benefit has been demonstrated by means of a case study. Furthermore, this case study has demonstrated that rules of thumb for service granularity often propagated in practice (e.g., Helbig and Scherdin 2008) must be seen critically. For example, a rule to design mostly coarse- or fine-grained services can lead to economically bad solutions. Reasons for this can be seen in disproportionately high implementation costs and possibly necessary multiple implementations of functionalities by different services. For an economic decision, it is necessary to analyze the given functionality graph and then to determine the adequate service granularity based on this specific graph. Hence, it may well be reasonable to implement the functionalities realizing a process by services with different granularities (see also the example in **Fig. 3**). The decision model and the software tool provide instruments to examine the economic effects. Furthermore, three mathematically defined metrics were presented and discussed that allow a comprehensible granularity assessment.

The economic analysis extends the domain analysis approaches to identify services. The compatibility to these approaches is primarily provided if the basic structure of the functionality graph can be derived from these approaches: For instance, Winkler (2007) suggests decomposing activity diagrams into functionalities and then arranging them so that functionalities which occur multiple times in different processes can be grouped. The result is a directed graph which is similar to our functionality graph presented. The decision about which functionalities should be implemented in a service is made argumentatively based on various assumptions in the approach by Winkler. Here, our model can explicate economic effects and provide additional decision support. Other approaches also conduct an analysis of the functionalities of a domain or a decomposition of functionalities as a basis for service identification (e.g., Fiege 2009; Offermann 2008). In these approaches, service candidates with different granularity can result according to the decomposition rule applied. Consequently, also in this context the presented decision model can be used after adapting it to the concrete approach.

Besides, some critical issues have to be discussed that define the need for further research: First, for the application of the decision model it is necessary to create a functionality graph which in turn is the result of a domain analysis – as illustrated in the case study of the financial services provider. This raises the question of whether and how, if necessary, a robust solution in terms of an adequate service granularity under economic aspects can be determined with an incomplete functionality graph (e.g., only a part of the functionality graph is modeled). This means that the solution identified should not change fundamentally in the course of an adjustment or completion of the functionality graph. However, it is obvious that a accurate model input has to be available for a well-founded decision making process. This holds also true concerning the input parameters of the objective function. Here again we must rely on quality-assured estimates. However, such a basis is also necessary without using a decision model if we want to make a serious estimation of the implementation costs of the project. Further research is needed here, which means the procedure for determining the linear and scaling factors has to be adapted especially for service development. Moreover, the sensitivity analysis – which focuses on imprecise estimates – needs to be enhanced. To be able to apply the decision model even with an incomplete functionality graph, further extensions appear helpful. For example, existing estimation methods might be adapted and integrated into the prototypical software tool and standard values for input parameters might be provided. In addition, future situations in which the current Functionality and Service Graph will be possibly modified have to be considered. Here, it is necessary, for instance, to determine probabilities for process or functionality changes (i.e., scenarios) and store them in the graph. Thus, future changes could be represented in a systematic way and for each scenario identified, for example, the expected value of the implementation costs can be calculated. These can in turn influence the choice of service granularity. The presented approach provides an appropriate starting point for all of these purposes.

References

Cf. German Version

Appendix 1: Verification of the feasibility of a solution

The following conditions for the feasibility of a solution have to be verified:

1) Every functionality m_j (with $m_j \in V \cup B$) must at least be assigned to one higher functionality or process (must be checked only once for the entire FSG)

$$\sum_i I_{m_i,mj} + \sum_i I_{p_i,mj} \geq 1 \qquad \forall \; j$$

2) For each solution that has been identified by a combinatory allocation, the following must hold: Select randomly a process *p'*. Then check every path starting with the edge *{p', m'}* $\in$ *E* whether *{s, m'}* $\in$ *E* holds for functionality *m'*.
   a) If this is the case, continue to the next path, i.e. edge *{p', m''}* $\in$ *E* etc. If there is no such edge, carry out the same procedure for every other process *p''*.
   b) If this is not the case, check for every path starting with edge *{m', m'''}* $\in$ *E* whether *{s, m'''}* $\in$ *E* holds for functionality *m'''*. If this is the case, check the next path with the edge *{m', m''''}* $\in$ *E.* If this is not the case, check all paths with the edge *{m''', m''''}* $\in$ *E* etc.
      - ⇨ If for a complete path *{p', m'} {m', m''}*, …, *{m''', m''''}*$\in$ *E* applies that *{s, m'}*, *{s, m''}*, …, *{s, m'''}*, *{s, m''''}* $\notin$ *E*, then the considered solution is not a feasible solution.

For every process *p'* it must be valid that all basic functionalities included in the process (these can be determined through the edges *{p', m}, {m, m'}* $\in$ *E*) are directly or indirectly implemented by services.

Appendix 2: Calculation of the total costs and the metric values in the example

For the service implementation shown in **Fig. 3**, the total costs of the implementation and the composition can be calculated as follows (maintenance and support costs are neglected in this example). It holds:

$$Z(I_{SM}) = C_R(I_{SM}) + C_K(I_{SM})$$

with matrix *I*I_{SM}*: S x M =*

$$I_{SxM} := \begin{array}{c|ccccccccccccccccccccc} & m_1 & m_{1.1} & m_{1.2} & m_2 & m_{2.1} & m_{2.1.1} & m_{2.1.2} & m_{2.1.3} & m_{2.1.2.1} & m_{2.1.2.2} & m_{2.1.2.3} & m_{2.1.2.2.1} & m_{2.2} & m_{2.2.1} & m_{2.2.2} & m_{2.2.3} & m_{2.2.4} & m_{2.2.4.1} & m_3 & m_{3.1} & m_{3.2} \\ S_1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ S_2 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ S_3 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ S_4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ S_5 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ S_6 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ S_7 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ S_8 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{array}$$

as service allocation (with $I_{s_i,m_j} = 1$ if functionality *m*m_j is implemented as service s_i, otherwise $I_{s_i,m_j} = 0$ holds). The following costs in MU result for implementing the services s_1 to s_8 based on the parameter values c_{var} = 3.0, c_{fix} = 90 and b = 1.075 using the term $c_R(s_i) = \sum_{j=1}^{|M|} I_{s_i,m_j} \cdot (c_{fix} + c_{\text{var}} \cdot (size_{m_j})^b)$:

Service	***s*** s_1	s_2	s_3	s_4	s_5	s_6	s_7	s_8
Costs $c_R(s_i)$	**7,877.88**	**5,126.41**	**56,736.64**	**5,126.41**	**10,700.32**	**4,052.26**	**8,999.51**	**15,324.08**

For example the costs $c_R(s_1)$ for implementing the service s_1 are calculated as follows:

$$c_R(s_1) = I_{s_1,m_{1.1}} \cdot (c_{fix} + c_{\text{var}} \cdot (size_{m_{1.1}})^b) = 1 \cdot (90 + 3 \cdot 1{,}500^{1.075}) = 7{,}877.88\ MU$$

Therefore, the implementation costs for all services s_1 to s_8 result to 113,943.51 MU.

Furthermore, the costs c_K for the service composition of the process „*Offering private loans over the Internet*“ has to be determined. This effort includes the size of the compositional logic that has not already been implemented by a service. As shown in **Fig. 3**, the composition logic of the functionalities m_1, m_2, $m_{2.2}$ and $m_{2.2.4}$ together with the process composition logic of 200 LOC is not implemented directly or indirectly by a service. Thus, the composition logic yields to a total of 1450 LOC. The composition costs for the process in the case study are calculated with the parameter values c_{var}^{comp} = 3.0 and f = 1.175:

$$c_K(m_p) = c_{\text{var}}^{comp} \cdot \left(size_{m_p}^{comp} + comp(I_{SM}, m_p)\right)^f = 3 \cdot (1{,}450)^{1.175} = 15{,}549.94\ MU$$

Summing up the costs for service implementation and composition, the solution shown in the above matrix I_{SM} have minimal total costs of 129,493 MU.

For the functionality and service graph presented in the case study (**Fig. 3**), the values for the three metrics can be calculated and interpreted as well. The following metric values for the distance-oriented metric, the scope-oriented metric, and the size-oriented metric result for the services s_1 to s_8:

Service	s_1	s_2	s_3	s_4	s_5	s_6	s_7	s_8	***Total***
Distance-oriented metric	**1**	**1**	**0.36**	**1**	**1**	**1**	**1**	**0**	**0.79**
Scope-oriented metric	**1**	**1**	**0.5**	**1**	**1**	**1**	**1**	**0**	**0.81**
Size-oriented metric	**0.44**	**0.79**	**0.44**	**0.94**	**0.88**	**0.95**	**0.90**	**0**	**0.67**

As the table shows, the value of 0.79 for the distance-oriented metric and the value of 0.81 for the scope-oriented metric are very similar. This is because six of eight services implement basic functionalities and thus hold a granularity of one (maximum fine-grained). Only service s_3 shows differences: Here, a value of 0.36 of the distance-oriented metric indicates that the service is implemented after approximately 1/3 of all paths between the process "*Offering private loans over the Internet*" and those basic functionalities which are indirectly implemented by service s_3. In contrast, the scope-oriented metric shows a value of 0.5. This means that service s_3 implements 50% of all functionalities that are part of the sub-graph (with the edge (m_p, m_2)). This means that if service s_3 would implement the preceding functionality m_2 "*Calculation and check loan application*" instead of functionality $m_{2.1}$ "*Perform scoring*", then the service doubles its scope of implemented functionalities and thus it would be maximum coarse-grained.
Additionally, also the values of the size-oriented metric are shown. They refer to the size of the functionalities measured in LOC. Overall, its value is slightly below the metric values of the other two metrics with 0.67. The reason is that with service s_1 a relatively large basic functionality "*Identify partners*" (in relation to the sub-graph with the edge (m_p, m_1)) is implemented.