EVOLUTIONARY DEVELOPMENT OF BUSINESS PROCESS CENTERED ARCHITECTURES USING COMPONENT TECHNOLOGIES

Asuman Sünbül
Kestrel Institute
Palo Alto, CA

Herbert Weber
Technical University Berlin
Computation and Information Structures
Berlin, Germany

Julia Padberg
Technical University Berlin
Theoretical Computer Science/Formal Specification Group
Berlin, Germany

The process centered paradigm changed the way of today's business organizations. Both the organizational structure and the IT-infrastructure are effected by this paradigm shift. For companies, competitiveness means meeting the continuously changing business requirements, concerning business environments *and* workflows. *Structural modifications caused by continuously changing business processes, and ad-hoc-modifications triggered by spontaneous events provoke adaptability problems: in all these cases, the software system must be adapted accordingly in order to be consistent with the modified business process. Because of the fact, that that adaptation is costly, time consuming, alternatives paradigms have to be considered. In this paper, we therefore propose to use an evolutionary strategy for the development of business process based applications. In our approach, each modification request caused by a change of the business process implies an evolution step in the architecture of the workflow based applications. In this paper we present* EVA*, a strategy for forming such* evolvable architectures *for communication and information infrastructure referred to as* EVA*.*[1] *The initial models for* EVA *are business process models as a basis for modeling system architectures. We use Petri nets for representing the dynamic characteristics of the business processes. The evolution concept on the software architectural level is covered by the application of component strategies. In this paper we will present a concept for the design of evolution in workflow components (Padberg et al., 1999) and explain the composition of these components.*

[1] This paper describes the conceptual foundations and design aspects of *EVA*. In contrast, (Weber et al., 2000) focuses on the formal foundations of *EVA*.

1. Software Architectural Evolution

Today, enterprises must react to continuously changing requirements caused by the constant changes of the market on which enterprises depend, changing society, and technology changes. In order to remain competitive enterprises must adapt their business processes accordingly. Workflow management systems (WMS) are often used to support the business processes of an enterprise, so that changes of these involve changes in the WMS, and subsequently "spontaneous modifications" of the software system. There exists two kinds of modifications:

- ***structural modifications,*** resulting from modifications of the business process itself and
- ***ad hoc modifications***, which are caused by spontaneous events during a business process.

More precisely, the following problems occur in the context of these modifications:

1. ***Changing basic structures***:
   Apart from the reorganization process, information and communication technologies are introduced into the enterprises in order to support the workflows. The technical infrastructure must be changed accordingly.
2. ***Integration of new components:***
   The constant modification of the business processes require the integration of new system components into existing software systems. In most cases, this means that already existing systems are combined with newly developed systems.
3. ***The adjustments of systems cause problems***:
   Continuously necessary adjustments introduce software problems. Often, the adjustment possibility of systems is not considered during the design conception time. This often results in the need for redevelopment into a new software system. Therefore, paradigms for the development and creation of adaptable, easily changeable software architectures, e.g. concepts for dynamically changing software architectures are needed.
4. ***Necessity for solutions in the area of evolutionary, scalable software architectures:***
   Instead of exchanging whole system parts in large intervals with high risks, we argue to create continuously maintainable systems. A step in this direction are reusable and exchangeable software components.

Software system have somehow an architecture but not all systems support architectural integrity, architectural consistency in structure, representation and documentation. The fact, that software systems are more often modified than written and the fact that the assumption defined above presume easy adaptability, modifiability and reusability, leads to the conclusion, that the common mainspring results mainly from the design stage of the software system.

As a conclusion, the *EVA* project aims at developing prototypes for an evolutionary software architecture. We argue, that software architectures are dynamic entities. Over time, systems need to be modified in response to changing requirements and changing business processes. Evolution support concerns both documentations as e.g. specifications, architectures, and other artifacts and the implementation of the system. The *EVA* model illustrates how continuously changing business-processes can be reflected by an evolutionary architecture, allowing ad-hoc modifications. Our evolutionary design principles follow beside the static and dynamic *EVA* component definitions also systematic methodologies and observance of invariance rules. In the sense of Continuous Software Engineering invariance means, that each step within a software life cycle has to follow certain rules. These rules are subject to apply throughout the life of a system. The following of invariance rules support the reusability of architectures.

2. Related Work

The Eva project aims to bring together the results of three research fields within software development in order to make software evolutionary: Component-based software engineering , software architecture and high-level Petri- nets are seen state of the art research mostly in parallel existence to each other:

Component based software development is a well accepted technology [(Goedicke et. al, 1991), (Medvidovic et al., 1999), Oreizy et al., 1998),(Anlauff and Sünbül, 1999)]. However, most architecture based approaches are not suitable for being used as design methodologies in the sense of the EVA approach, because they do not provide abstraction mechanisms that are necessary for designing system architectures. On the other hand, component technologies such as JavaBeans (Sun Microsystems, 1997), DCOM/ActiveX (Sessions, 1997) or CORBA(Siegel, 1999) either don't solve composition problems (e.g. semantic conformity, consistency, or compliance with constraints and models), because they are implementation techniques. Furthermore, an evolutionary design of the system can be achieved, if the connections between these components are also treated as first class design entities on the same level as components. This issue is neither supported by architecture based approaches nor any of the technologies mentioned above. By making software architectures flexible, which means that we don't construct conglomerations and wild wiring between components, we expect to better support application evolution and flexibility. In this case, the target system is open for future changes. A practical composition concept must accommodate the use of existing and heterogeneous component applications. Therefore, in this paper we propose the EVA approach, which supports this kind of developments.

FUNSOFT nets are high level Petri-nets which have been particularly developed to support software process modeling. Their semantics is defined by Predicate/Transition nets. That enables to benefit from standard analysis techniques approved for Predicate/Transition nets. In our approach, we use Feva-Nets, a subset of FUNSOFT nets for a domain specific use.

3. Design of Business Process Centered Architectures

We understand business processes as sequences of activities with a certain start and endpoint. The flow of the activities is controlled by so called *actors*. Business processes are further characterized as consisting of structured and unstructured parts. The structured parts are predictable and planable activities while the unstructured parts are unpredictable, spontaneous events. The predictable parts can also be seen as the static parts of the system, the unpredictable parts as the dynamic ones.

In order to model a business-process oriented architecture based on process models as dynamic parts, we propose the following concepts:

- For modeling business processes we use special high-level Petri-Nets, so called *Feva-Nets* offering structured and formalized descriptions of workflows. *Feva-Nets* are characterized by the use of temporal logic, different switching modes and typing of the places [(Sünbül, 2001), (Padberg et al., 1999)]. The tokens of Feva-Nets carry value and are typed using a rich type system. Feva-Nets have a flexible and adaptable firing behavior, and are tailored to model business processes in business applications. A more detailed description of Feva-Nets can be found in (Weber et al., 2000).

- As the basic design model, we propose an *evolutionary component based architecture*. This component model is referred to as *EVA*, supporting the idea of reusability and exchangeability of system parts.

The paper is organized as follows: First we will introduce the *EVA* component model, then Feva-Nets are described. Section 3 illustrates our approach using a "call center'' as a case study.

4. The EVA Component Model

Evolutionary design requires a clear separation between interdependencies and intradependencies of the involved processes. This leads to the conclusion, that the intradependencies are best modeled with component technologies. The intra dependencies are modeled with Petri-Nets, more detailed views will be presented in the sequel.

We understand components as data capsules providing services. Plugs are given by interfaces, which can be used from outside to access these services. Components are durable in that sense, that they can exist independently from specific software development or execution environments. This property requires components to be self-contained. Further, components should be assembled dynamically to form applications. They must inter-operate according to a set of rules (Goedicke et al., 1991).

The component model of EVA distinguishes between *Business Process Components*, *Business Object Components* and *Code Components*. Each component consists of an *Export Interface* describing the services that are provided by the component, an *Import Interface* describing the services being required by the component, and a *Body* containing the realization of the services listed in the export interface. The export interface may additionally contain a list of constraints specifying the conditions on how to use the exported services. The difference between code components and business objects lies on a conceptual level: Code components represent basic functionality that is used in many different location. Other than business object components, the services provided by code components can be seen as library functions; business object components are closely related to a single aspect and/or concept of the company's applications. In case of business process components there are two specialties:

The body of a business process component contains *Feva-Nets* specifying the dynamic behavior of the services provided by the component. Feva-Nets are high-level Petri-Nets, and therefore, a petri-Net interpreter is also attached to each business process component in order to execute these nets.

In addition to the signatures of the required services, the import interface of the business process components may contain an import net describing how the imported services are used within the component.

The overall system is given by an instance of the class "Business Process" consisting of exactly one business process component providing one service. The net contained in this component describes the business process as a whole on an abstract level and will be refered to as "toplevel-net".

The constraints contained in the export interface of a component are given in term of temporal logic formuli. These constraints usually define dynamic aspects of the services, as in the following example:

```
S1 (int)(int)
S2 (int, int)(string)
always(S2 ⇒  once S1)
```

The exported services in this case are S1 and S2; S1 takes an integer value as input parameter and returns an integer value, while S2 takes two integer values as input parameters and returns a string. The constraints expresses, that, before S2 can be invoked, the service S1 must have been called at least once. In general, the service signatures of export and import interfaces consist of the name of the service, the parameter list for the input parameters, and the parameter list of the output parameters. Additionally, the behaviour of the service wrt. the assignment of values to the output parameters:

All each output parameter contains a defined value of corresponding type the service returns; this is the default case;

DET: only one of the output parameter contains a defined value, the others are undefined;

Complex: a combination of ***All*** and ***DET***

In order to efficiently support the design of workflow based systems, *EVA* distinguishes between three categories of software components occurring in an enterprise.

- enterprise-specific components realizing special "in-house" functions that correspond to the basic entities of the enterprise's business as, for example service requests, costs, and orders;
- branch of industry-specific components realizing more general functionality like general accounting software, program development systems etc. In most cases, these components are purchased from outside vendors.
- process-oriented components realizing the business processes occurring in an enterprise. These components are usually part of workflow management systems.

According to these categories, the *EVA* model distinguishes between three kinds of components: business object components for the first, business process components for the second and code components for the third category, where business object components represent a special kind of code components. In the *EVA* model the whole software system is given as a single *business process component* containing the basic structure of the enterprise's business process. This *business process component* will be referred to as "top-level net". The different kinds of *EVA* component are described in the following in more detail. The *EVA* model does not exclude the use of externally developed components, but these components have to follow some rules and guidelines. At least the provided/required services and constraints on these have to be visible for the developer.

5. Business Process Components

Business process components are used to model the dynamic processes occurring in a system. In its body, a *business process component* contains a Feva-Net describing the realization of the component. Feva-Nets are a specialized form of *FUNSOFT* nets (see [(Gruhn, 1991), (Deiters and Gruhn, 1998)]) focusing on those features being necessary to make the *EVA* concept work. Feva-Nets can be transformed to place/transition net (see (Reisig, 1985)). Hence, they can be easily checked for the validity of temporal logical formulas. See (Weber et al., 2000) for a more detailed description of Feva-Nets. The signature of ***S*** and Feva-Net ***N*** specified in body correspond in the following way:

- for each input parameter of ***S*** there exists exactly one place in ***N*** that is marked as input place;
- for each output parameter of ***S*** there exists exactly one place in ***N*** that is marked as output place;
- depending on the behavior of ***S (DET, ALL)*** the service terminates if all output places are marked ***(ALL)***, or only one ***(DET)***

Furthermore, the body of a business process component contains a Petri-Net interpreter (PNI) being responsible for executing the specified petri net.

The import interface of a business process component may contain an ***import Feva-Net*** being used as an abstraction of the processes using the imported services. In general, the import net represents an abstract view of the net contained in the body of a business process component and focuses on the relation of the imported services while the net in the body of the component may contain additional transitions needed for the realization of the business process. More precisely, if S_B is the set of symbols used in the body Feva-Net N_B, and S_I the set of symbols used in the import Feva-Net N_I, then $S_1 \subseteq S_B$. Furthermore, if P is a predicate defined on symbols $X \subseteq S_1$ and the import net N_I, written as $P(N_I,X)$, then is $P(N_I,X) = P(N_B,X)$, meaning that whenever a predicate holds in the import net, it also holds in the body net.

The import nets are used to automatically check the consistency of component compositions: the constraints given in the export interfaces of the imported services can be checked against the use of these services in the business process components. See (Weber et al., 2000) for a more formal presentation of this feature.

6. Business Objects

The OMG specification for interoperability defines a business object component as an accessible object that is associated with an identifiable, real-world entity such as a customer or order, this definition reflects also our understanding as follows: A business object component is the IS representation, from requirements analysis through deployment and run-time, of an autonomous" business concept or process. It consists of all the software artifacts necessary to express, implement and deploy the given autonomous business concept as an equally autonomous, reusable elements of a larger information systems.

According to this definition, a business object component stays a coherent unit throughout the development life cycle emerging at the end as a set of software components that can be independently evolved while business requirements change and develop. The question why we regard business object components as a separate unit is the fact, that there are causal dependencies between business processes and software: we argue, that software and business processes are two different entities, but both are changing (evolving) over time, and these changes are somehow linked with causal influences in both directions. While designing the system software, one should consider, that a modification of the business process should not lead to a modification of the system. The main issue of business object component is, that it separates business logic from resources. This enables e.g. database schemas and technologies to change without necessarily affecting business logic. We can state, that business object component form the invariant parts of business processes.

As mentioned above, business objects represent the "static" parts of the company's business. From a technical point of view, these business objects are regarded as objects in the sense of object-oriented programming and modeling concepts. In EVA, each of these objects is encapsulated in a separate component where the export interface is given by the specification of the public methods of the object. In contrast to other components regarded in EVA business objects carry an own state being reserved throughout the life of the object. In EVA, the business objects are regarded as data structures and are used to assign values and types to the services exported and imported by the other kind of components. In particular, that means, that the tokens of Feva-Nets carry business objects as their values and are typed accordingly. Consider, for example the net in Figure 1 expresses that a service S expects a business object of type T1as input parameter and returns a business object of type T2 and type T3. The token in the input place of S represents that the concrete business object x1 is accessible as input parameter. The methods defined in T1 are available in the realization of S using the input parameter x1; additional services from other business objects must be explicitly imported, for example for accessing instances of T2 and T3.

7. Composition Concepts

In order to build a complete software system, the individual parts of the components must be joined according to their service use. The resulting compositional structure must be hierarchical and cycle-free (directed, acyclic graph). The renouncement of cycles is necessarily due to the import and export interface check of the temporal-logic and the other conditions. For example, the import interface of a component can contain "import nets". These import nets can be used to automatically check whether a given business process component fits in the import interface of the component containing the import net.

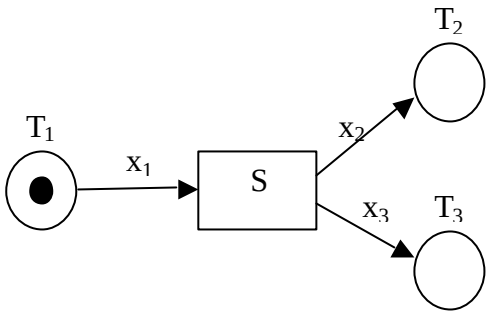

Fig. 1 Using typed business objects as tokens in Feva-Nets.

That means, that in this special case the composition of components can be fully automated. The mathematical formalisms were regarded in detail in (Padberg et al., 1999).

The composition concept presented in (Sünbül, 2001) can be used to describe the composition of *EVA* components. Here the composition of components consists of two aspects that need to be verified:

- Does the signatures and the specifications of the imported services fit to the import interface of the importing component?
- Are the constraints of the exporting component fulfilled by the importing component?

In case of a business process component, the services of the import interface are given by the transitions of the import net. Each transition represents an import service; the incoming edges represent the input parameter and the outgoing edges represent the output parameter of the service. Assuming, that the net in Figure 1 is the import net of some component *A*, then the corresponding import service of *A* would be

$$S(\text{in } x_1{:}T_1, \text{out } x_2{:}T_2, \text{out } x_3{:}T_3)$$

As one can see, the "types" of the places are used as parameter types of the import method.

In *EVA*, the requirements for export services are expressed by means of Feva-Nets; the constraints of the use of export services of a given component are specified using temporal logic. A typical example of such a constraint is the fact that an initialization service has to be called prior to any call to another service of a component. If a another component imports services from a component with this kind of constraint, the import net of the importing component can be used to check whether the constraint is actually met.

The information needed for the composition is given in the export section of the component. If a component is subject to be merged into new system, it should be ensured, that the component service parameters are compatible with those of the system parameters. Besides these conditions, the call sequence of the service calls, which are specified in *EVA* with temporal logic, have to be checked. Both is to be checked automatically at compile time, that a run-time check can be avoided.

In (Weber, et al., 2000), the aspect of component composition especially for business process components is described in more detail.

8. Case Study: A Calling Center

We illustrate our concept using the case study of a "calling center" of a telephone company. We have chosen this case study, because it reflects the typical scenarios of many enterprises, e.g. customer care

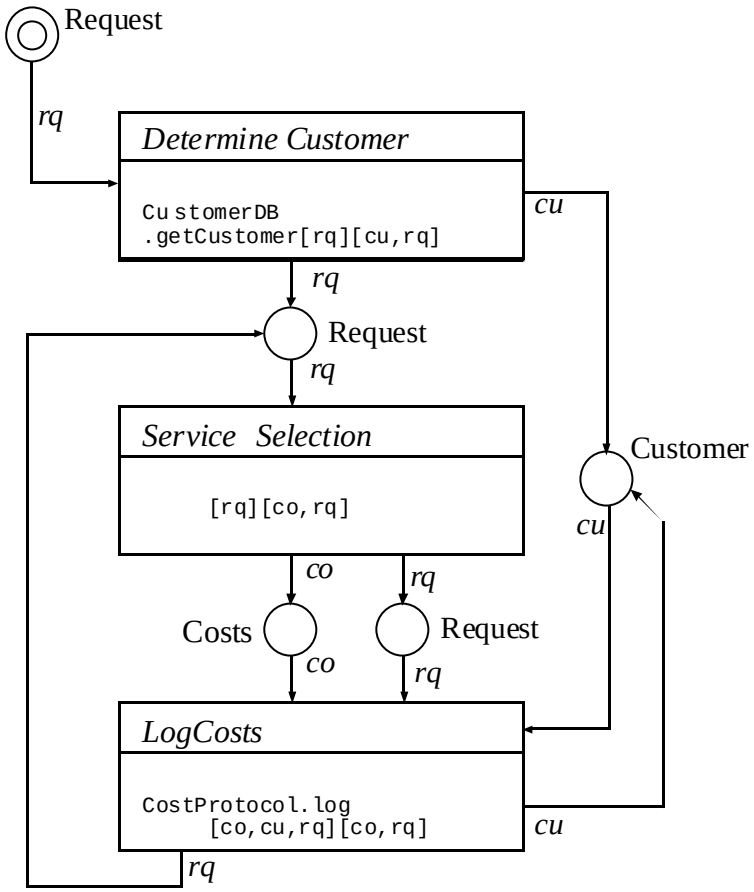

Fig. 2 Toplevel net of the calling center.

and billing companies, telephone service companies, medical or technical emergency centers, police/fire departments etc.

In our scenario, a customer of the calling center submits a service request which must be processed within a short time period. In our case study, we will distinguish between two kinds of customer requests: requests for ordering services and request for getting information about the current status of his/her data stored in the data base of the calling center.

9. High-level Representation of the Calling Center

As mentioned in the previous section, the system itself is given by a top-level net, the transitions of which are then refined by more detailed. The top-level net of the calling center is shown in Figure 2**.** The top-level net describes how one request is processed by the system. This view represents at the same time the architectural structure of the whole system: “DetermineCustomer”. “ServiceSelection”, and “LogCosts” are sub-components connected with input/output arrow, which can be viewed as high-level connectors. The request itself can consist of many sub-requests that must be processed one after the other. The leftmost place represents the input place of the net. In the first step, the *Determine Customer*

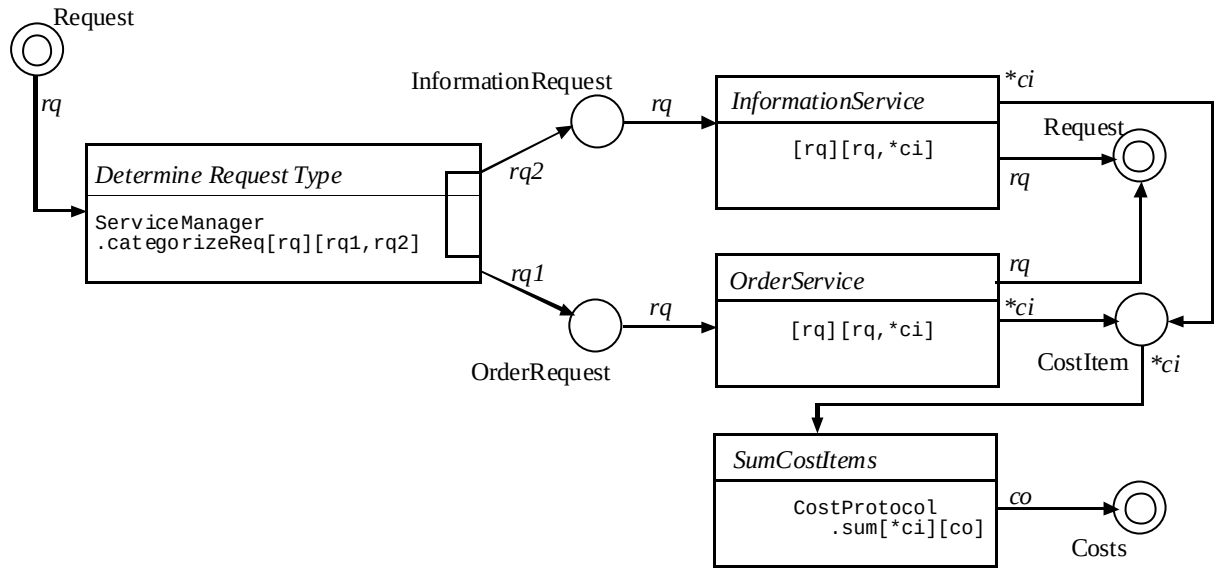

Fig. 3 The Business Process Component "Service Selection.

transition, information about the customer who submitted the request is queried from the customer data base of the calling center. This is done using the *getCustomer* service of the *CustomerDB* code component. As mentioned above, the labels of the edges represent business objects; their types are given by the label of the corresponding place. For example, *cu* is a business object of type *Customer* and *rq* is a business object of type *Request*. The *Service Selection* transition further analyses the request, executes it and returns the computed costs for the request. This transition will be refined to a separate sub-net being explained below. The transition *LogCosts* collects the information about the requests and logs the costs for it using the *log* service of the code component *CostProtocol.* If the request contains further parts, these will be processed one after the other which is expressed by the outgoing arrows from the *Log Costs* transition.

10. The Business Process Component for selecting services

The business process component *Service Selection* is used in the top-level net for the determination of the kind of request and for calculation of the cost for the request dependent on the type. The net is given in Figure 3. The net has one input place and two output places which are marked by the doubled circle. This corresponds to the representation of the component in the high-level net. The request is first categorized using a service of the code component *Service Manager*. The corresponding transition is of type *DET* meaning that only one of the output places is filled with an business object token rather than both as it would be the case for conventional transitions. Depending on the category either the *Information Service* or the *OrderService* is used to calculate the costs of the service. These transitions are refined as *business process components*; their description is not contained in this paper and can be found in (Padberg et al., 1999). The notation **ci* means that a list of cost items is given as output of the service components and forms the input to the *SumCostItems* transition being responsible for calculating the costs of the services using the *sum* service of the SumCost Items code component.

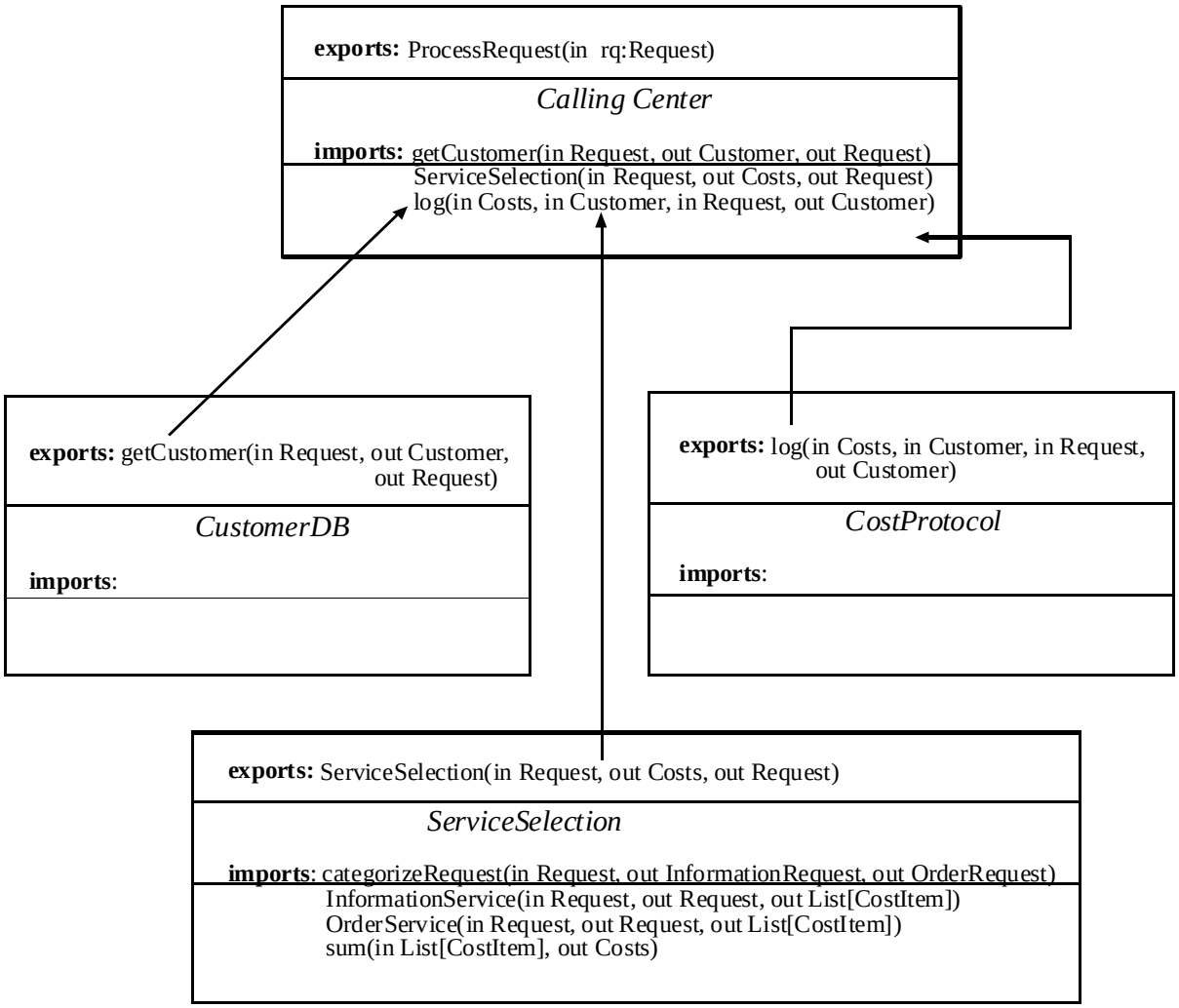

Fig. 4 Component diagram of the calling center example.

Figure 4 contains an overview of the components involved in the calling center example. In this figure, only the signatures of the components are regarded; the arrows represent the connectors from the export services to the import services.

11. Evolutionary Change

We want to consider now the case, that a new requirement concerning the toplevel business process is imposed on the system: Checking whether the customer is blacklisted, and if yes, deny any access to the services of the calling center. This evolutionary change can be modeled by simply adding to the toplevel Feva-Net (Figure 2) a transition representing this additional check. The concrete way of how this is done is not important in this case, it is obvious, that these kind of evolutionary changes can directly be modeled using the Feva-Nets and are thus immediately realized in the system's architecture without changing other parts of the system.

12. Conclusions

An *EVA* based system is composed from individual components, in order to enable high evolutionability of the system. This enables, that in case of modifications or exchange of system parts, only individual components are involved, wide major parts of the system remain unaffected. In order to ensure a high reusability of the components, we have defined three types of components. All these components have the equivalent structure to the outside, but they differ in realization. To better help support evolution we propose to structure also the communication between the components. Structurally regarded, there are two kinds of communication possibilities in *EVA*, namely *vertical* and horizontal communication. The horizontal communication can only be done by using a business object component.

This paper illustrates how components are composed using the *EVA* approach: Business objects are used as invariant parts of the system and represent tokens in the nets, while the *business process components* represent those parts of the system who are subject to frequent changes. Evolutionability can be described as the ability to anticipate the locations of changes in a system. For industrial companies that means that a set of system parts are more or less invariant, like for example the customer or the request, while other might change quite frequently, like the workflow within the company.

13. References

Anlauff, M. and Sünbül, A., 1999, "Domain-Specific Languages in Software Architecture". *Integrated Design and Process Technology (IDPT)*.

Deiters, W. and Gruhn, V., 1998. "Process management in practice - applying the FUNSOFT net approach to large scale processes". *In Automated Software Engineering*.

Deiters, W, Gruhn, V., and Weber, H.,1994. "Software process evolution in MELMAC". In Daniel E. Cooke, editor, *The Impact of CASE on the Software Development Life Cycle*. World Scientific, Series on Software Engineering and Knowledge Engineering.

Ernst, M., Cockrell, J., Griswold, W.G., and Notkin, D., 1999. "Dynamically discovering likely program invariants to support program evolution". In *Proceedings of the 21st International Conference on Software Engineering*, pages 213-225. ACM Press.

Goedicke, M., Schumann, H., and Cramer, J., 1991. "On the specification of software components". In Jean- Pierre Finance, editor, *Proceedings of the 6th International Workshop on Software Specification and Design*, pages 166-174, Como, Italy. IEEE Computer Society Press.

Gruhn, V., 1991, "Validation and verification of software process models". *Lecture Notes in Computer Science*, Vol. 509.

Harrison R., Sheppard M., and Daly, J.W.,1997, "Process modeling and empirical studies of software evolution". In *Proceedings of the 19th International Conference on Software Engineering (ICSE '97)*, pages 675-675, Springer Verlag.

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

Muller, H.A.,1993. "Experimental software engineering should concentrate on software evolution". *Lecture Notes in Computer Science*, 706.

Neighbors, J., 1992. "The evolution from software components to domain analysis". *International Journal of Software Engineering & Knowledge Engineering*, 2(3):325-354.

Oreizy, P., Medvidovic, N., and Taylor, R., 1998. "Architecture-based runtime software evolution". *In Proceedings of the 20th International Conference on Software Engineering, pages 177-186*. IEEE Computer Society Press.

Padberg, J., Sünbül A., and Weber, H., 1999, "EVA: Evolutionsfähige Architektur für Kommunitions- und Informations-Infrastrukturen". *Technical report, Technical University Berlin*, 1999.

Reisig, W.,1985. "Petri Nets". Springer Verlag.

Sessions, R., 1997. "COM and DCOM: Microsoft's Vision Distributed Objects". John Wiley & Son.

Siegel, J., 1999, "Component and object technology: A preview of CORBA 3". Computer 32,5 (May 1999), 114-116

Sun Microsystems, 1997. "JavaBeans™", Graham Hamilton (ed.) Version 1.0.1

Sünbül, A., 2001, "Architectural design of evolutionary software systems in Continuous Software Engineering". PhD thesis, TU-Berlin.

Ward, M. and Bennett. K., 1995, "Formal methods to aid the evolution of software". *International Journal of Software Engineering and Knowledge Engineering, 5(1):25-47*.

Weber, H., 1999, "Continuous engineering of information and communication infrastructures". *Lecture Notes in Computer Science, 1577:22-29*.

Weber, H., Padberg, J., and Sünbül, A., 2000, "Petri net based components for evolvable architectures". *IDPT*.