Active-passive hybrid data collection

Raf Haesen[1,2], Lotte De Rore[1], Monique Snoeck[1], Wilfried Lemahieu[1], Stephan Poelmans[2]
[1] LIRIS Group, Faculty of Economics and Applied Economics, K.U.Leuven
Naamsestraat 69, 3000 Leuven, Belgium
{raf.haesen, lotte.derore, monique.snoeck, wilfried.lemahieu}@econ.kuleuven.be
[2] Campus Vlekho
Koningsstraat 336, 1030 Brussel, Belgium
spoelman@vlekho.wenk.be

Introduction

This pattern guides the design of software components that exchange large amounts of data in order to deliver a certain decision service. The general objective is to minimize the total (i.e. development, maintenance and runtime) cost of the system to be built. After the introduction of a set of basic design scenarios to gather data, the pattern proposes an optimal combination of those scenarios such that the total cost is minimized.

Name

Active-passive hybrid data collection

Audience

System designers, Software architects

Context

This pattern is applicable in each situation where a complex decision process has to be executed based on a large list of input parameters. As such at least two different component types are identified: the first component, called the **provider**, processes the large collection of data elements in order to make the decision. The other component is the one that interacts with the end-user and the one that triggers the provider to make the decision; that component is called the **consumer**.

The input parameters for the decision process originate from typical sources in an enterprise such as databases, individual files or interactive input by the end user. The output of the process is assumed to be elementary, e.g. the positive or negative outcome of a decision.

Examples of a decision process with many inputs and one straightforward output are:

- Is a person entitled to child benefit, given detailed data about the domestic situation and on the employment of the parents?
- Can the customer get a loan, given his past, current and expected future financial situation?

- Is it safe to insure an object of a customer, given detailed information about the person and the object?

The latter question will be worked out as running example throughout the remaining sections. The example is extensively elaborated in the following section.

Example

Consider the real-life case in which a consumer wants his house to be insured together with the house content. The supporting application to be designed has to fulfill the following requirements:

- The application needs to accept or reject the request and needs to calculate the insurance premium in case of acceptance. This yields a two-staged approach. The acceptance step investigates whether or not to accept the request for insurance. The second step is the tarification step which generates a price offer. The first step requires a substantial amount of data in order to evaluate all possible reasons for rejection. On the other hand, a minimum of data may be sufficient to provide the customer with a first, rough estimation of the price, purely for informational purposes.
- The person who wants to have an object insured can be an existing or a new customer of the insurance company. For an existing customer some data will be available in a database while all data about a new customer will have to be retrieved by interactively questioning the customer.
- The application checks whether the customer is present on a blacklist of untrusted payers that is externally available to the insurance company.
- Before the insurance request is accepted, the application checks for the absence of fraudulent family members.
- The premium of the house content depends on the fact whether the customer possesses exclusive and expensive goods, such as jewelry or special stamp collections. The premium increases proportionally to the value of those possessions.
- The data used to calculate the premium for the house content can be altered after the construction of the application. For example the premium for a stamp collection may initially only depend on the number of stamps in the collection. After examining past insurance claims, the insurance company may wish to consider also the exact sort of stamps for the premium calculation.

Problem

Many applications require the transfer of large amounts of data between components. The responsibility of collecting the required data can be assigned to several components, for example to the provider who uses the data to make a decision or to the consumer who calls the provider. The actual distribution of data collection across the service consumer and the service provider has a huge impact on the total cost of the system. If only the consumer collects all data before passing it on to the provider, the consumer needs to be modified each time the provider needs other data. Hence, it might seem better to let the provider collect all necessary input. However in certain

situations the provider doesn't have the capabilities to gather all data, for example because some data is not available to the provider.

Given the fact that a lot of data has to be collected, the designer faces the problem of figuring out how to distribute the responsibility of collecting data across the provider and the consumer. So we need to infer the optimal degree of activity of the provider: an active provider searches itself for missing inputs while a passive provider gets all necessary data supplied together with the call. Additionally some of the data may already be available in a database, whereas other parts must be queried from the user.

Forces

It is appropriate to divide the forces in two groups: forces that directly influence the cost of the development and maintenance of the system, and forces that occur during runtime. The former contribute to the fixed cost of the solution while the latter contribute to the variable cost.

Development forces:

- *Heterogeneous data properties:* the system should support different possibilities concerning the data to be collected. This force can be subdivided in:
    - *data availability:* On the one hand some data will be requested through the user interface of the consumer, such as data about a new customer, while on the other hand some of this data may already be available in a database, such as data about an existing customer. Clearly, the system needs to deal with the fact that data will be available from different sources. This force expresses an absolutely necessary condition for the system to be useful.
    - *data visibility:* in some cases it may be desirable to hide which data is used for taking the decision. Clearly, that information should not be collected by the consumer. For example the insurance company may wish to compare the customer profile to a database of known cases of fraud. If one doesn't want to divulge the use of this kind of data in the decision process, then the provider should be responsible for collecting this data.
    - *data accessibility:* access restrictions may be different for each component. For example: some data may be available in a database but in order to protect the privacy of the customer only the consumers may have access rights to that information.
- *Interface modification:* the propagation of modifications due to the modification of the interface of the provider should be minimized. It is generally understood for components not to depend on a component of which the interface regularly changes. Since the transfer of data between components is discussed, possible interface changes are the addition, modification and deletion of data elements.
- *Support of multiple kinds of consumers:* one provider can deliver its service to multiple consumers with heterogeneous capabilities. In that case, the provider should be able to deal with consumers with different capabilities.

- *User-friendliness:* it is indispensable to avoid collecting data via the user interface that is already available in any other way. The manual retrieval of data that already exists puts an unnecessary burden on the end user and the customer who is redundantly questioned.

Runtime forces:

- *Database connections:* the number of database connections for both the provider and the consumer should be as small as possible. It is cheaper to query a database once to collect all the data than to open several sequential connections to the database to retrieve the same amount of data.
- *Transferred data volumes:* the size of data transfers between the components should be as small as small as possible, in particular if both components reside on different network locations. Especially large volumes of data that are only used in exceptional cases should only be transferred when needed. Clearly this force impacts both the volumes of database query results and the volumes of network transfers.

Solution

The solution is elaborated in three phases: first four basic scenarios for data collection and data access are introduced. Subsequently the impact and resolution of all forces is identified for each scenario. Although each basic scenario will have some interesting properties, it will not solve the problem on its own. Therefore an integrated solution is constructed which combines the four basic scenarios so as to balance all forces in order to minimize both development and runtime cost.

1. Basic Scenarios

There are two aspects to consider: which component takes the initiative to collect the data and where does the required data reside? The consumer can be responsible for the data collection. On the one hand the consumer can collect data via its interface (and a human actor), for example a bank clerk who fills in the properties of an object to be insured while interacting with a customer. On the other hand the consumer can query a database to collect for example all known data about a customer. The provider can also have capabilities to gather the data during the decision process. It can ask for missing data to the consumer, for example additional information might be necessary when insuring exceptionally expensive jewelry. Finally the provider can also access databases or individual files.

Four basic design alternatives can be constructed based on these fundamental ways to collect data. Since each alternative represents a simplistic way of accessing and collecting data they form no solution on their own. Cleary the “data availability” force states that the system needs to handle data that is distributed over different sources. It will be shown that only a balanced combination of all scenarios yields a satisfying result.

1.1. Passive-provider without stored data

Context:
For this scenario, we assume that all data is available from the end-user only. There is no stored data and no external data.

Solution:
In the first passive-provider scenario the consumer collects all data *via the user interface* (the request for data is consistently represented by a thick line). Subsequently the consumer calls the provider (the call initiative is consistently represented by a thin line) and all collected data is sent (the data flow is consistently represented by a dashed line) to the provider at the same time. This passive-provider scenario is represented in figure 1.

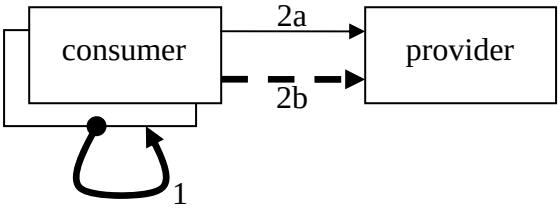

Figure 1 – Passive-provider scenario with data collection via the user interface

Forces:
Data availability: All the data is requested through the user interface and if we assume that the end-user has knowledge of all required information, this scenario scores well on this force.
Data visibility: In this scenario there is no possibility of hiding data that is used in the decision process: the consumer needs to know all the data used in the decision process as it is the only component collecting data.
Data accessibility: Data access rights are usually given in advance and determine to what data a component has access to. One can assume that the consumer will have the necessary rights to access most customer-related data. There is no guarantee that the user has access to or is willing to provide information that is obtained externally, such as fraud history data. Usually, the use of external data will be subject to specific restrictions and hence it is more likely that this type of data is going to be accessible by the decision-taking component rather than by the consumers.
Interface modification: The passive-provider scenarios yields a "fat" interface between the consumer and the provider. As a consequence, the consumer needs to be modified each time the provider interface changes.
Support of multiple consumers: Because there is a "fat" interface between consumer and provider, it is impossible to keep the interface generic and cater for several (types of) consumers.
User-friendliness: If the same user wishes to re-submit a same request, all data will have to be input a second time. As there is no possibility to store data, this scenario scores badly on user-friendliness. In practice, it is only feasible in case of a very small number of data elements.
Database connections: There are no database connections in this scenario.
Transferred data volumes: The volume of transferred data is maximal in this scenario since all data has to be collected first and subsequently transferred at each request. Also data that is

potentially not required for the particular decision case has to be included in the transfer. As a result, this scenario scores badly on this force.

In summary, this scenario scores badly on too many forces and can only be viable in situations were a very limited amount of data is required for the decision.

1.2. Passive-provider with stored data

Context:
In this scenario no interaction with the end-user is possible or desired. All data is available in a number of databases, accessible by the consumer.

Solution:
In this second passive-provider scenario (see figure 2) the consumer first *queries a number of databases* to collect all the required data. In a second step this data is passed on to the provider who calculates the result.

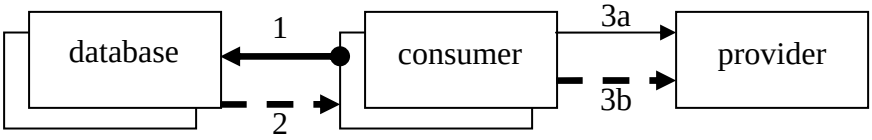

Figure 2 – Passive-provider scenario with database access

Forces:
Data availability: All the data is retrieved from existing databases. Therefore, there is an availability problem with data that has not (yet) been entered into a database, e.g. data about a new customer.
Data visibility: As in the previous scenario, there is no possibility of hiding part of the data that is used in the decision process.
Data accessibility: Also this force is addressed in a similar way as in the previous scenario. One can assume that the consumer will have access to most customer-related data. However, there is no guarantee that it also has access to information that is obtained externally, such as evidence of previous fraud.
Interface modification: Also this scenario yields a fat interface and each time the provider changes his interface, the consumer will need to be adapted as well.
Support of multiple consumers: Because there is a “fat” interface between consumer and provider, it is impossible to keep the interface generic and cater for several (types of) consumers.
User-friendliness: Since there is no interaction with the user, this scenario is inherently user-friendly: it avoids situations where a user has to provide information multiple times or has to provide information which was already present in a database.
Database connections: All needed databases are accessed only once. Because repeated database access is avoided, this scenario scores reasonably well with respect to this force.

Transferred data volumes: Again, all data has to be transferred at each request, also data that is potentially not required for the particular decision case. As a result, this scenario scores badly on this force.

In summary, this scenario scores badly on too many forces and can only be viable in the very rigid situation where each decision case requires the same set of data, which in addition should already be readily available in a database.

1.3. Active-provider with callbacks

Context:
For this scenario, we still assume that all data is available from the consumer only, either by means of interaction with the end-user or from one or multiple databases.

Solution:
Figure 3 represents the first active-provider scenario where the consumer calls the provider while passing only a minimum of data. The provider requests the required information from the consumer. This *callback mechanism* is represented by a thick arrow from the provider to the consumer and a resulting data flow in the opposite direction. Here we make abstraction of the way the consumer collects the data, i.e. via the user interface or by means of database queries. With a callback mechanism all data can be retrieved at once or in several steps where each step depends on the current stage in the decision process.

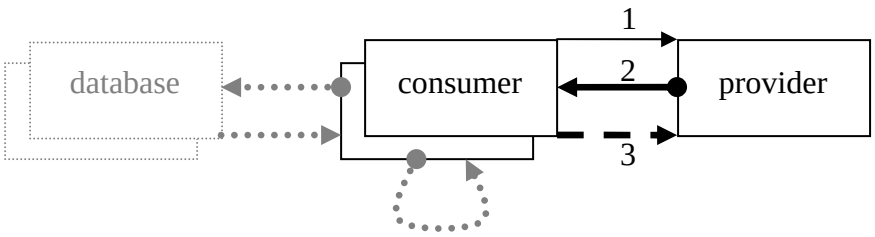

Figure 3 – Active-provider scenario with callbacks

Forces:
Data availability: All the data is requested through either the user interface or existing databases. If we assume that the end-user has knowledge of all required information, this scenario scores well on this force.
Data visibility: Again, because the provider requests the necessary data from the consumer, it is still not possible to hide part of the data that is used in the decision process.
Data accessibility: Again, one can assume that the consumer (upon request by the provider) will have access to most customer-related data, either from a database or through the user interface. However, there is no guarantee that it also has access to information that is obtained externally.
Interface modification: Compared to the previous scenarios, this one improves the resolution of this force: because not all data is passed upon invocation, the interface can be kept “thin” and

generic. If the callback mechanism is implemented by means of a generic interface as well, the dependency between the consumer and the provider's interface can be kept low.
Support of multiple consumers: A "thin" interface and generic callback mechanism facilitate the inclusion of different types of consumers.
User-friendliness: In this scenario, user-friendliness depends on how the consumer collects the data requested by the provider: the use of a database will be more user friendly than collecting all data through the user interface, which holds the risk of requesting the same data multiple times.
Database connections: It is possible for the consumer to have no database access in the active-provider scenario with callbacks if data is retrieved via the user interface. Since on the other hand data can be requested during different steps in the decision process, several sequential database accesses may be required.
Transferred data volumes: Transferred data volumes can be minimal if, through the callback mechanism, only effectively required data is requested. On average however, data transfer will still be substantial if a lot of data is needed for the average decision case.

In summary, this scenario improves the resolution of a number of forces. Still, there remain major problems with visibility and access to external data. This scenario provides a candidate solution in cases where no external data sources to which the consumer had no access rights are required and where all used data can be disclosed to the consumer. When implementing this scenario, one will also need to take care to limit the number of database connections.

1.4. Active-provider with stored data

Context:
In this scenario, we assume that all data is available to the provider. The provider has no means of interacting with the end-user.

Solution:
The last design in figure 4 represents an active-provider scenario in which the provider seeks for missing data *by accessing databases*. Again the consumer triggers the provider while passing only a minimum of data and the provider has the responsibility to gather the required data. Also in this scenario the required data can depend on the case at hand.

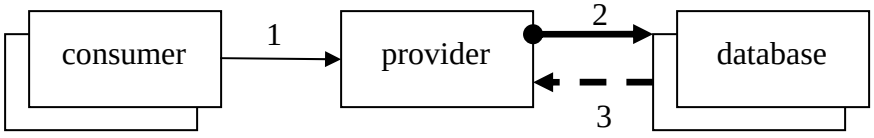

Figure 4 – Active-provider scenario with database access

Forces:
Data availability: The active-provider scenario with database access is the worst-case scenario for this force because all consumers have a "hidden dependency" on the data that is queried for by the provider. If the required data is not available in the database, this scenario offers no

possibility to retrieve it from the consumer. Assume for example the insurance company implements this scenario and a new customer wants to insure a particular object. The provider most probably needs information about the new customer, but that information is not yet available in the database.
Data visibility: This scenario is the only one that resolves this force: if the provider wants to hide which data is used for taking a decision, then obviously it cannot delegate the data collection to the consumer but must collect the data itself.
Data accessibility: Because the provider is responsible for all data collection, this force is problematic if customer data is protected and can only be accessed with explicit permission by the consumer. On the other hand, the use of external data will be subject to other restrictions and it is more likely that this type of data is going to be accessible by the decision-taking component rather than by the consumers.
Interface modification: As in the previous scenario, only a small set of data is passed upon the initial invocation. Therefore, the interface can be kept "thin" and generic, hence reducing the consumer's dependency.
Support of multiple consumers: Same as above, a "thin" and generic interface facilitates the inclusion of different types of consumers.
User-Friendliness: This scenario is very user friendly, because no data is collected through the user interface.
Database connections: Database access is performed by the provider. Since data can be requested during different steps in the decision process, several sequential database accesses may be required.
Transferred data volumes: This scenario scores the best on this force: transferred data volumes between consumer and provider can be minimal, because only a limited set of data is passed in order to initiate the request. After the initiating call no more data needs to be transferred between the consumer and provider.

Although this scenario resolves forces that remained unresolved in previous scenarios (data visibility and transferred data volumes) it scores too badly on too many other forces to be a realistic solution. Especially data availability and access rights to user-related data will turn out to be problematic in almost any case.

2. Evaluation of the basic scenarios

The influences of all forces for each basic scenario are summarized in table 1. From this overview, we conclude that a realistic solution needs to combine the advantages of each scenario to resolve as many forces as possible. In particular, data collection by the consumer is required to cater for data availability and consumer data access rights, whereas the active provider with database access is required to cater for the possibility of hiding confidential data used in the decision process. Limited modification propagation, support for multiple consumers and limiting the transferred data volume are best dealt with by the active scenarios.

Forces → Scenarios ↓	Data availability	Data visibility	Customer data access rights	External data access rights	Modification propagation	Multiple consumers	User-friendliness	Database connections	Data volumes
Passive-provider without stored data	++	--	++	??	--	--	--	++	--
Passive-provider with stored data	+-	--	++	??	--	--	++	+-	--
Active-provider with callbacks	+-	--	++	??	++	++	+-	++ or --	+-
Active-provider with stored data	--	++	??	++	++	++	++	+-	++

Table 1 – Evaluation of forces for each scenario

3. Integrated solution

The key to the solution of the given problem is to combine all basic scenarios in one hybrid scenario. As such the hybrid scenario consists of two active and two passive parts as represented in Figure 5. Part 1 and part 2 represent the passive-provider scenarios: part 1 represents the data queried from a database and part 2 is retrieved via the user interface. Part 3 and part 4 represent data that is actively collected by the provider. Part 3 is retrieved from the consumer via a callback mechanism and part 4 is retrieved from a database.

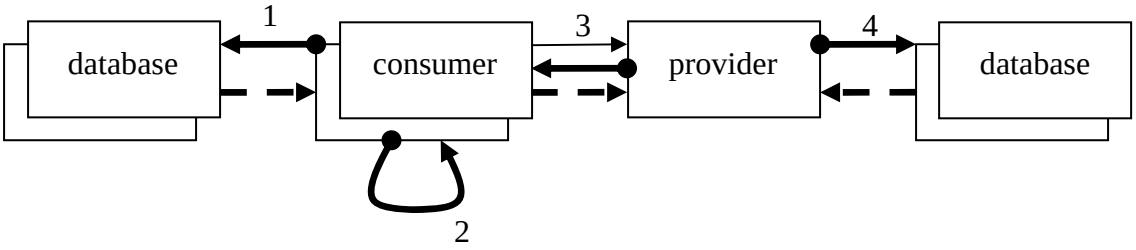

Figure 5 – Hybrid scenario

All possible data is subdivided and assigned to one of the four parts in the hybrid scenario. The subdivision of the data depends on the properties of the data under consideration. Figure 6 shows a decision tree representing which data has to be assigned to what part of the hybrid scenario.

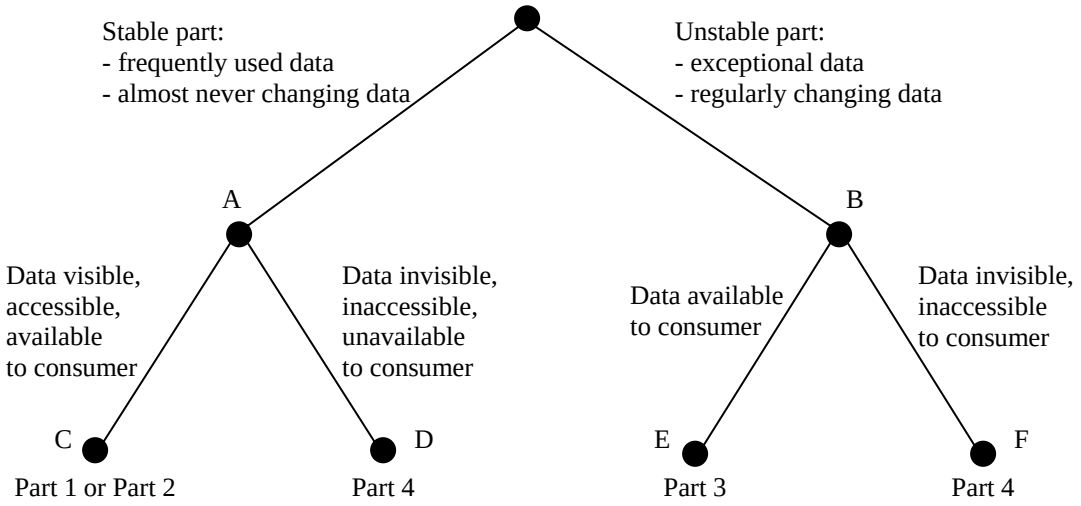

Figure 6 – Decision tree

At the first level, a distinction based on the frequency of use and frequency of change of data is made. Data that is used in exceptional cases or data that regularly changes has to be retrieved in an active way. Data changes cover both changes to the data itself and changes in the need for data. As such sources of interface modification are isolated and propagations of modification are minimized. At the same time the multiple consumer support force is resolved.

The data that is used in practically every case is assumed to be stable, i.e. that part of the data will change only very exceptionally. Clearly there is no risk for frequent interface modification and multiple consumers can easily be supported.

The second level resolves the heterogeneous data force and also decides which part of the data has to be assigned to which part of the final scenario. Stable data that is visible, accessible and available to the consumer should be retrieved in a passive way (node C). The choice between retrieval via databases or via the user interface comes down to a balancing between the user-friendliness force on one hand and the database connections force on the other hand. The other part of the stable data that is unavailable, inaccessible or invisible to the consumer must be retrieved by the provider via database access (node D). Clearly the user-friendliness force is resolved but database access cannot be avoided.

As already mentioned the provider should retrieve unstable data in an active way (node B), i.e. via callbacks from the consumer or from a database. Clearly exceptional data will most likely be available to the consumer and should be requested via the callback mechanism (node E). Again a trade-off between user-friendliness and the number of database connections must be made. Finally, some unstable data can be inaccessible or invisible to the consumer and must be retrieved

via database access. This way the user-friendliness force is resolved and database access cannot be avoided (node F).

Example resolved

We applied the pattern to the previously given example. Using the decision tree discussed in the previous section, the data extracted from each requirement is assigned to one of the four parts of the hybrid scenario represented in Figure 5.

- Stable data concerning existing customers should be retrieved by the consumer via database access. (part 1)
- Data about new customers can only be retrieved via the user interface of the consumer. (part 2)
- The description of expensive items should be actively requested by the provider via the callback mechanism. (part 3)
- Information about blacklisted customers or fraudulent family members should be queried from a database by the provider. (part 4)

Finally we show that most forces are resolved when retrieving the data this way.

- The requirement to use the application for acceptation as well as for price calculation means that *different types of consumers should be supported*. If a quick price estimate for a new customer has to be calculated, the provider cannot actively retrieve data from a database ("hidden data dependency"). Consequently passive data retrieval via the user interface must be supported for new customers. Stable data for existing customers should be requested from a database in order to create a *user-friendly* solution.
- The verification of a person being present on an external blacklist is an example of *data (in)accessibility*. Since the blacklist will only be accessible to the provider, the latter should actively query the list.
- It will be undesirable to ask the customer for the existence of fraudulent family members or even to mention this rule. This illustrates *data that is to be kept invisible* to the consumer and should as such only be actively retrieved by the provider via database access.
- The expensive possessions of a customer denote *data that is only available to the consumer*. As mentioned earlier the provider should actively request exceptional data from the consumer in order to *minimize the transferred data volumes*.
- The fact that some required data will be modified is an argument to retrieve that data in an active way by the provider in order to *minimize the propagation of these modifications*. Data that is *available to the consumer* (e.g. stamp collection size) should be retrieved via callbacks while data only *available to the provider* (e.g. volatile legislation matters) should be queried from a database.

Acknowledgements

The authors would like to thank Neil Harrison for his very useful remarks during the shepherding process and several people from KBC for proposing this interesting case. This pattern has been written as part of the KBC-Vlekho-K.U.Leuven research chair on ‘Service and Component Based Development’ sponsored by KBC Banking & Insurance.