Digital Business Ecosystem

Contract n° 507953

Work Package 15 DBE Business Modelling Language

Task B22 - Walkthroughs

D13.5 Walkthrough Report

Information Society Technologies

Project funded by the European Community under the "Information Society Technology" Programme

Contract Number: 507953

Project Acronym: DBE

Title: Digital Business Ecosystem

Deliverable N°: D13.5

Due date: 31.7.2006

Delivery Date: 22.2.2007

Short Description: Walkthrough Report

Partners owning: FZI

Partners contributed: FZI

Made available to: Public

Versioning		
Version	**Date**	**Name, organization**
1.0	**8.9.2005**	**Tim Romberg, FZI**
2.0	**22.2.2007**	**Tim Romberg, FZI**

Quality check

Peter Stanbridge, IBM

Contents

1 Introduction

The goal of this document is to showcase some of the practical B2B application capabilities of the DBE infrastructure. The intended audience are business analyists, software architects and other evaluators with some technical background.

1.1 Scope of walkthroughs

The DBE infrastructure consists of many individual modules, in various states of maturity. Some of them have very generic character, for example the FADA technology. Therefore it is hard to pin down any kind of application that could not use some part of the DBE infrastructure. But with limited resources, not all of these application can receive the same level of support from infrastructure developers, and the infrastructure can only be engineered according to so many quality criteria. As an example, although it would be possible in theory to route voice data through FADA nodes in real time, it is probably not the best technology out there for building a VoIP network.

Business services and software services

The word "service" is often used in association with the DBE infrastructure. It is important to distinguish between the concept of a "software service" and the concept of a "business service" or "real-world service".

A **business service** is something that a company offers for money; traditionally the "services" sector has been distinguished from the "industrial", but there seems to be a trend towards integrating the two, so that a majority of companies delivers services of some kind.[1] So for simplicity, all of these will be called a "business service" in the context of DBE.

According to the intention and scope of the project formulated at its outset, the DBE infrastructure should support the interaction between companies (specifically SMEs). This means it does not try to address business-to-consumer (B2C) relationships and transactions, like for example Amazon.com. It is, however, not limited either to only supporting seller-buyer/service provider-client-relationships.

As the DBE infrastructure is a piece of information technology, it is potentially useful only for the information and communication aspect of a given business transaction.

[1] HP, for example, offers printer management and maintenance services along with its printer hardware. Telephone companies deliver specialized hardware to their clients for network access.

A **"software service"** is a technical concept created to help modelling large software systems. Relationships between components of an application are defined as a client-server relationship under this model. (This should not be confused with the physical client-server architecture where "server" denotes an actual machine.) The server software component offers the software service, which has a well-defined interface and specification as well as an implementation. The details of the implementation are unknown to developers of client software components, which creates the desired flexibility in the architecture. The server component runs independently of actual requests. Clients can appear and disappear at any time, and initiate conversations with the server component. This bears a metaphorical resemblance to the business relationship between a service provider and a client. The software service specification and interface correspond, metaphorically, to a contract in the business environment. This architecture has become the dominant model for business applications in recent years, although other architectures, like data-centred integration (enterprise data models, Data Warehouses) or the pipes-and-filters architecture are still used in special areas.

In order to support a given business scenario, a typical constellation is for service providers to offer a set of software services for their clients; for example a catalogue or a booking service. But there are other possibilities, where it's the (business) client who sets up a software service for suppliers to access (e.g. to forward demand projections to them). A drugstore chain in Germany, for example, has set up an extensive reporting service based on a Data Warehouse where its suppliers, companies like Procter & Gamble, can analyze sales of their products almost in real time (up to the previous day), down to individual shops.

1.2 Challenges in B2B applications

Well-known functions of B2B applications are:

- Supplier directory (who delivers what)
- Catalogue management (exchange of product and service catalogue)
- Product classification (shared numbering, classification and attribute schemes)
- Product configuration
- Ordering, Delivery, Billing, Payment
- Request for Proposal (RfP), Request for Quotation (RfQ) – see the conference organiser case, for example
- Product returns and reimbursements
- Rebates

We can roughly differentiate between applications which address the *deep integration* between two partners who work together on a long-term basis and those

that support *ad-hoc interactions*. A supplier directory, for example, is a typical ad-hoc application. The other applications in the list are typical for ad-hoc interactions on market places, online shops or supplier portals, but will also frequently provide the basis of a deep integration between two partners, which will go far beyond these applications to include long-term activities like joint engineering, marketing, and planning.

1.3 Scenarios discovered during the use case analysis

The use case study of the tourism and conference organiser opportunity spaces has revealed the following, more unusual challenges, some of which could be typical for business ecosystem with a high degree of networking between SMEs:

Service aggregation

Service aggregation is performed for example by a travel agency as the core of its business. Hotels and sports shops can provide it as an added service.

The aggregation involves several activities, such as

- Informing about the existence of or even recommending the added services
- Booking added services
- Informing the other service providers about the client's characteristics
- Collecting payments on behalf of the other service providers

Demand aggregation

In the tourism domain, there are events like, for example, hiking trips, that can only take place with a minimal number of participants. When an individual client contact does not book enough people, service providers look for other participants and try to find a convenient date for everybody. Needless to say, this process can be complex and time-consuming, and could benefit from being accessible to all participants through a web page.

Shared client records or profiles

Both the service aggregation and demand aggregation needs could possibly be met with the help of a shared client record or profile. In the case of service aggregation, it would avoid having to enter the client's name, address and other characteristics several times (into the backend systems of each service provider). In the demand aggregation case, there could be a centrally managed calendar of all events during the client's stay; it would also be easier for the client to access all information concerning the entire stay through one interface. The concept could be carried further towards "collaborative CRM", where several service providers collaborate in managing the client relationship. However, this kind of sharing might not be legal under some consumer privacy legislations, or be rejected by clients if they are given a choice.

In particular, the following table shows the individual scenarios discovered in the Aragon region, classified into supplier-related and customers-related scenearios:

Supplier related	Customer related
Procurement and inventory management (Viajes Oriente, Hotel Hospedaría, Food & Handicraft Gift Shop) Supplier coordination and service aggregation Manual, on behalf of customer • Viajes Oriente • Equipo Barrabes (including route planning) Automatic • Forwarding of dependent orders - hotel cleaning services • Payment forwarding • Catalogue aggregation Exchange of client profiles, joint CRM (Hotel Hospedaría)	Client profiling, CRM Viajes Oriente, Hotel Hospedaría ("attract longer staying customers") Client aggregation (min. # of participants, scheduling; Equipo Barrabes, Escuela de Parapente) Online Booking, Shopping, Order Tracking (Radical Snowboard, Food & handicraft gift shop) Online Catalogue Browsing Discovery of service providers

Table 1: Applications supporting tourism scenarios in the Aragon region

The collaboration between hotel and providers of different sports activities is especially interesting:

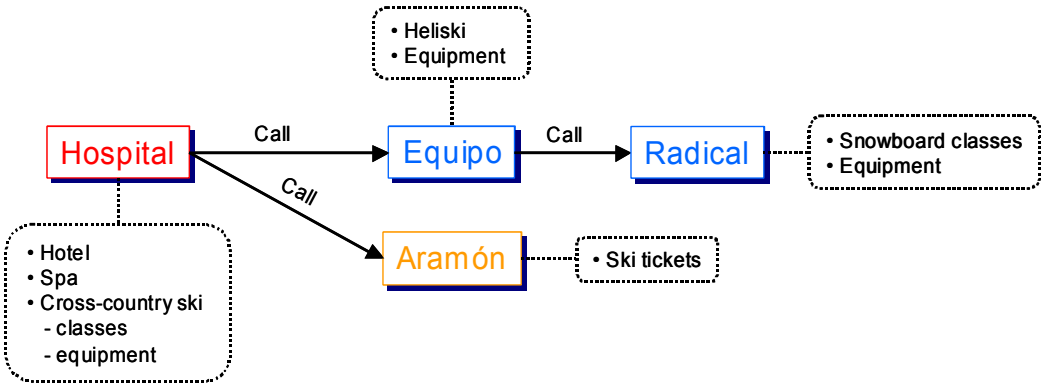


Figure 1: Collaboration of hotel and sports activities providers (figure 7 "booking channels" in use case report)

In this constellation, the hotel ("hospital") acts as the central contact point for the customer. In other scenarios, it is the travel agency Viaje Oriente.

The use case analysis on Conference Organization in *Tampere, Finland* has shown quite similar processes; for example, the conference organizer (eBRC) will usually reserve a sufficient number of hotel rooms in several hotels. Then these hotel rooms are offered to the participants through the conference web site (run by SuviSoft). It is therefore a case of service and catalogue aggregation.

To further classify the scenario applications above, one can use the distinction between Transaction, Collaboration and Information[2]. Each of these categories defines the kind of relationship between the objectives and actions of participants in the real world and the data exchanged:

Transaction-oriented applications document a real-world business transaction, e.g. an order or a payment, by a data record exchanged between partners. This data record is then typically archived and unchanged – although inside each partner's backend system, a transaction will often trigger a change in existing data, e.g. data representing warehouse stock levels, or representing the other partner.

Collaborative applications support participants in their joint work on some object which can be represented in data, for example a text document or an engineering blueprint. The participants share this representation. The data exchanged between the participants represents changing versions or updates of the object.

Information-oriented applications is what we call the vast area of applications where the data exchanged serves neither of the purposes mentioned above, for example, an online catalogue or a weather forecast service. It would be easy to further subdivide this area.

Very roughly, although not always, companies have been looking for ways to automate their business *transactions* first before adopting the other two categories of business applications.

Each category also poses unique technological challenges.

The above applications could be classified as follows:

[2] Refer to http://en.wikipedia.org/wiki/Collaborative_software

	Supplier related	Customer related
Transaction-oriented	Procurement and inventory management Supplier coordination and service aggregation Manual forwarding of orders Automatic forwarding of orders Payment forwarding	Client aggregation (min. # of participants, scheduling) Online Booking, Shopping, Order Tracking
Collaborative	Joint planning Yearly tourism forecasts Event/tour planning (route planning, etc.) Joint special offers ("Easter special", etc.)	Event/tour planning
Information-oriented	Service aggregation Catalogue aggregation Exchange of client profiles, joint CRM (Hotel Hospedaría)	Client profiling, CRM Online Catalogue Browsing Discovery of service providers

Table 2: Classification of tourism and conference organization applications

In the area of collaborative applications, new planning applications have been added which are mentioned as being a necessary part of other activities (done previously or during the process).

The following chapters treat B2B applications from a slightly more technical and abstract point of view, in order to develop the patterns needed to implement them. They address the following forms of interaction:

- Catalogues – this topic needs to be addressed in almost all B2B applications. From an abstract technical point of view, a catalogue is a data collection accessed in a read-only fashion, usually with no changes during one interaction.
- Transactions – orders, payments, and so on.
- Collaboration – some shared object (like a calendar, a customer profile, or a project plan) that business partners access together.

2 Catalogues

Advertising and describing the products and services one is offering is a likely first step in any B2B scenario, be it product- or service-related. The classic B2B application, established in the product area for this is catalogue management. There are several standards in this area:

- Standard file formats for catalogues:
  - BMEcat (open German standard),
  - cXML (proprietary standard by Ariba)
- Standards for product identification:
  - GTIN (Global Trade Identification Number, formerly EAN = European Article Number),
  - ISBN (International standard book number, subset of GTIN)
- Standards for product classification:
  - UNSPSC,
  - GPC,
  - eClass[3],
  - RNTD (RosettaNet)

Each of them has a considerable following, but none of them is really established across industries and countries. Probably the most widely used format for catalogues today, at least among SMEs, is CSV (comma separated values), conveniently editable in Microsoft Excel and other applications.

The DBE infrastructure does not propose any specific way to implement catalogues (perhaps, although it was designed with services is mind, it should propose one, in order to provide a default interface for existing applications and standards in the product area).

However, the DBE Infrastructure contains a component for a global multi-vendor catalogue of products and services – the Semantic Service Registry. The Semantic Service Registry serves as a Business Directory or "Yellow Pages" service, where information about available products and services can be looked up without contacting individual providers. The Directory allows much more level of detail, and more sophisticated structure than the typical "Yellow Pages", but it is more limited in size and up-to-dateness than a typical full online catalogue.

[3] Officially written „eCl@ss", since it was created during the brief period at the end of the last century when using the "@" (at-sign) instead of "a" was very fashionable. For easier reading, we shall not.

The following examples show how the full function of a catalogue can be implemented as a combination of the Semantic Service Registry with an online catalogue service provided through a DBE service interface.

	Semantic Service Registry	**Catalogue service**
Distributed?	Yes	No
Availability	Very high	Depends on provider
Size of provider-specific data	Limited	Unlimited
Up-to-date?	Limited to individual publications (a few per year)	Always up to date
Semantics	Usually and preferably shared with other providers Standard Query User Interface available from the DBE Infrastructure	Shared or provider-specific Query User Interface must be implemented

Table 3: Comparison of Semantic Service Registry and Catalogue Service

2.1 Computing equipment shop example

2.1.1 Structure of product catalogues

An online shop is certainly the most classic example for a catalogue service. Shops for computing equipment typically carry several hundred to several thousand items. Customers are often interested in detailed technical information about an item before purchasing it (they may also base their decisions on reviews published by magazines or other customers, but we will not treat this aspect here).

This technical information can often be conveniently expressed in the form of property lists, which allow comparing several items according to selected properties. For example, DVD burners could be compared by their burning speed, their support of various media types, and their connector type.

2.1.2 BML for product catalogues

With BML, it is possible to define a schema for this type of information:

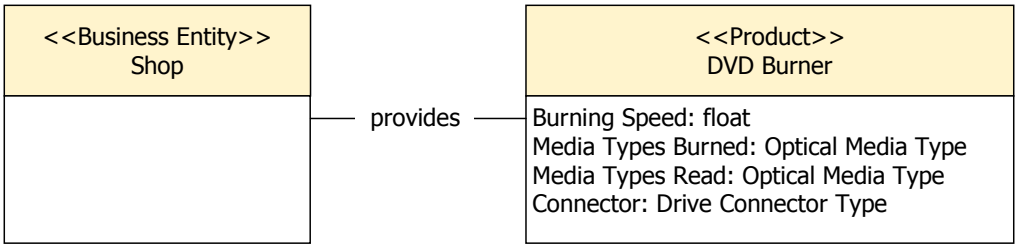


Figure 2: BML for DVD Burner catalogue properties

In this case, the burning speed has been defined as an attribute of type float (representation of a Real number, such as 2.5 which could be interpreted as 2.5x

speed). The other attributes refer to concepts in an Ontology, which also defines the allowed values (instances of the concept). For Optical Media Type, these would for example be DVD+R, DVD+RW, DVD-R, DVD-RW, CD-R and CD-RW.

BML as implemented today reveals some limitations with respect to typical product catalogue metamodels; for example, the Media Types Burned attribute should allow for a list (or set) of media types, but we cannot specify this. Also, in a typical catalogue or product classification scheme, we would be able to define certain attributes on the level of a broader category (such as Optical drives in general or DVD drives in general), and the more specific category of DVD burner would inherit these attributes. The current BML implementation does not support inheritance. Many product category schemes let attributes have an identity independent from a specific category they are used in, so they can be introduced at several nodes in the product hierarchy. In BML, attributes are defined as belonging to one class.

On the other hand, BML allows reusing concepts from a common Ontology, such as Optical Media Type and Connector Type.

2.1.3 DVD burners in product classification standards

For comparison, the GPC classification standard[4] defines a "brick" CD/DVD Drives for Reading and Writing in the following hierarchy:

SEGMENT : 65000000 - Computing

CLASS : 65010300 - Computer Drives

BRICK : 10001127 - CD/DVD Drives - Reading/Writing

DEFINITION : Includes any products that can be described/observed as a device that slots into or attaches externally to a computer, designed to play and create CDs or DVDs. Excludes products such as CD and DVD Read-Only Drives.

BRICK : 10001128 - CD/DVD Drives - Reading Only

…

GPC then only defines two attributes for this brick:

ATTRIBUTE TYPE : 20001033 - Type of CD/DVD Drive Reading/Writing

ATTRIBUTE TYPE : 20002018 - Integrated/External

In contrast, the eClass standard classifies as follows:

19 Information, communication, and media technology

19-03 Accessories for Hardware and Peripheral Devices

19-03-02 Drives (PC)

19-03-02-05 DVD burner (PC)

19-03-02-06 Combination drives (PC)

[4] http://www.gs1.org

It is interesting to see how different these two classifications are, with the latter introducing a high-level category for PC-drives specifically, and separating between CD, DVD and combination drives.

eClass also defines many more attributes (properties) for DVD burners, currently they are:

BAE887001 - Acceptance standard
BAE862001 - Audio output (Y/N)
BAF068001 - Data transfer rate
BAE156001 - Dimensions
BAF178001 - External and internal transmission speed.
BAF050001 - interface for CD and DVD
BAG322001 - Max. buffer size of intermediate storage
BAF731001 - mean access time
BAF047001 - Memory capacity of formatted CD and DVD
BAF723001 - Min. processor design
BAF719001 - Min. working storage
BAF482001 - Number of media
BAF166001 - Number of req. connections and outputs
BAF850001 - Product certification (Y/N)
BAG017001 - Service (Y/N)
BAG053001 - Software included (Y/N)
BAF049001 - Speed of reading from CD and DVD
BAF052001 - Speed of writing to CD and DVD
BAE712001 - Total number of available connections
BAF051001 - Type of CD and DVD
BAE711001 - Type of connection
BAG120001 - Type of current supply
BAF973001 - Type of interface
BAF703001 - Type of medium
BAA261001 - Type of support
BAF340001 - With guarantee (Y/N)

We would expect an online shop to provide a level of detail closer to the eClass property set, or even greater detail.

2.1.4 Choosing which part will be advertised in BML

The scenarios we would like to support are:

- Finding a vendor who sells electronics equipment, and searching for DVD Combo drives with a certain speed and interface;
- Aggregating the catalogue of several vendors into one catalogue (e.g. for a local retail shop, or a corporate purchasing portal); searching for the DVD drives in this aggregate catalogue; comparing prices

A choice has to be made as to which information will be advertised as BML, and which in the catalogue. This choice will be made based on the technical characteristics in Table 3. The following figure shows a spectrum with the most durable and low-volume data on the left, and the more volatile and high-volume data on the right:

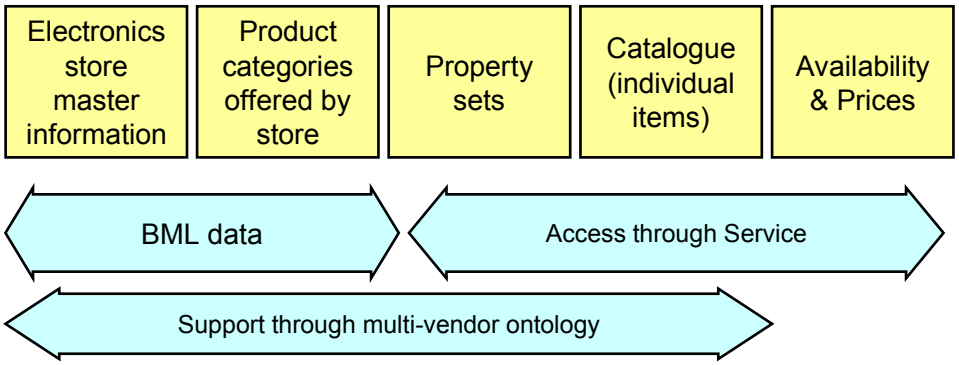


Figure 3: Use of BML for computing equipment catalogue

The electronics shop's master information refers to such things as its name and location. This will certainly be published as BML. We must also publish the product categories offered by the shop if we want to be able to select vendors based on them. In principle, this could be done by publishing each category as a BML class, like shown in Figure 2 for DVD burners, in the company's specific BML model (M1). The properties of this class would then represent the product categories' properties. However, this only makes sense if the individual items of the category are then published in the BML manifest (M0 data). There are two reasons not to do this (cf. Table 3):

- The catalogue, even without availability and prices, consists of rather high-volume and quickly changing data;
- As each electronics shop sells a slightly different set of product categories, and perhaps uses slightly different property sets in their catalogue, each shop would end up with an M1 model of its own. It will be very hard to use standard search to access any of these specific model elements in a search across stores, or to implement services which access them; chances are any shop-specific M1 model elements will be ignored.

Therefore we suggest a different model, which can be used by all electronics stores.

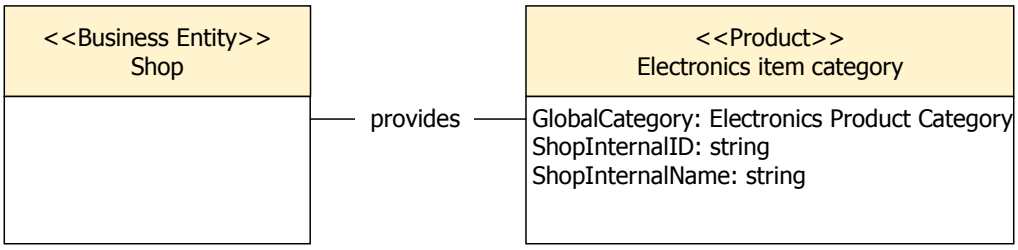


Figure 4: Generalized BML Model for Electronics shops

The M1 model contains only one product class, the "electronics item category". It is used to describe not individual items, but item categories available at the shop. Electronics product category is a shared concept in the ontology; the electronics community at large can manage the common product categories there; for example, category names for a variety of natural languages can be kept in the ontology, so that each shop does not have to provide translations. In the

BML manifest, the global categories in the ontology can be mapped to internal Ids and Names for these categories. If a category offered by the shop does not exist yet in the ontology, at least information about its name in the shop's language is available.

This BML model will therefore allow querying all suppliers of DVD drives in the Semantic Service Registry. The subsequent interaction goes through the service interface.

It will usually be possible to list all available item categories through the service interface as well:

```
Category[] getCategories();
```
[5]

with the `Category` class representing essentially the same information as the BML Electronics item category class above, perhaps with some additional runtime information, such as the number of items in that category. We could also imagine that the categorization in the individual shop's catalogue is more fine-grained; several internal categories ("DVD read-only drives, DVD burners, …") could map to one public category ("DVD drive"). As a next step, we should be able to query the property sets of each category:

```
Property[] getProperties(string categoryInternalId);
```

where each Property contains the name of the property, the data type, the allowed values (in case of an enumeration), and so on. It may also refer to an instance of a concept Electronics Item Property in the industry ontology, which may in turn mirror the properties defined in an existing classification standard.

If we follow the logic of the existing classification standards, there is an *n:m* relationship between categories and properties, which would be more appropriately represented by the following interface:

```
string[] getPropertyIds(string categoryInternalId);
Property getProperty(string propertyId);
Property[] getAllProperties();
```

We (the Client, that is) now know that the shop sells DVD burners, and that DVD burners in its catalogue have properties such as Data Transfer Rate, Types of optical media read, types written, DVD reading speed, DVD burning speed, and so on. What we don't know yet is which drives are actually offered and available. This is the next step:

```
CatalogueItem[] getItems(string categoryInternalId);
```

Each catalogue item should ideally be a complex object, as shown here

[5] At the time of writing, the SDL implementation had some limitations regarding the passing of objects and object lists as parameters and return values. It would be tedious to describe the workarounds to this each time, and we expect the problem to be addressed shortly. `Type[]` will stand for a list of items of type `Type`. In Java, the most likely equivalent is the interface `List<Type>`. In all interface descriptions and code examples, purely syntactic elements like the `public` keyword for declarations are omitted.

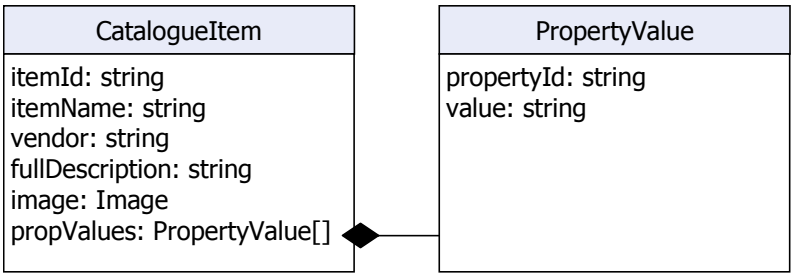


Figure 5: Catalogue item with property values (value object in service interface)

If the SDL interface does not allow the definition of such complex return values, including arrays and binary objects (image), this structure could be serialized into an XML document and deserialized into the same structure on the client.

Remember that the "conceptually clean" alternative of defining a `DvdBurner` class in the interface, with all its catalogue properties as attributes, is completely impractical, since it ties any implementation to a particular product category of a particular shop at a particular time. This neither makes sense on the server side nor on the client side.

In many B2B systems the catalogue functions are separated from functions accessing the current availability and, in the case of electronics, price, since the latter is highly volatile information which is usually stored in a different system. (For other industries, prices may be more stable, and thus part of the catalogue). Following this separation, we would have:

```
Availability getAvailability(itemId);
Currency getPrice(itemId);
```

where `Availability` could represent some enumeration of availability states, and `Currency` is some number type used for representing amounts of money.

This is all the information we need in order to find the DVD drive we want.

Here is the summary of the catalogue interface:

```
Category[] getCategories();
string[] getPropertyIds(string categoryInternalId);
Property getProperty(string propertyId);
Property[] getAllProperties();
CatalogueItem[] getItems(string categoryInternalId);
Availability getAvailability(itemId);
Currency getPrice(itemId);
```

2.1.5 Aggregating shops

We now look at the common challenge of aggregating the catalogues of several existing shops (wholesalers) into one catalogue; for example for a local retail shop or a corporate purchasing portal. A similar challenge is faced by the price comparison portals widely present on the Internet today. We will assume that neither of these manages their own inventory. (If they do, it can be treated more or less like another wholesaler.)

First of all, creating a BML manifest to represent the aggregated store will be easy. The BML model (M1) remains the same. The BML manifest (M0) of the aggregated shop simply has to include as product categories the superset of the product categories of the individual shops. Here we can see an important benefit of managing the product categories in a common ontology: It will ensure that the equivalent categories (e.g. DVD drives) in all individual shops are in fact aggregated into one category of the aggregated shop. Imagine the difficulty of doing this with a pure ID or name-based approach! Although probably, our aggregation service will still have to do some mapping itself, since there may be shop-specific categories which need to be matched to each other.

Unfortunately, as of yet, there is no tool for automating this first aggregation step.

For the service implementation, there are two basic architectures: a virtualized approach, and a replicated approach.

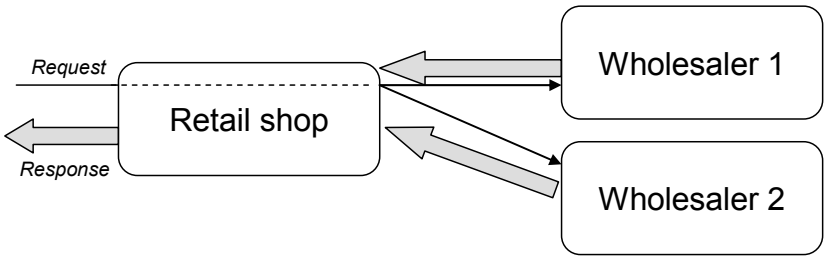


Figure 6: Virtualized approach to shop aggregation

In the virtualized approach, each request to the retail shop's catalogue spawns a corresponding request to each relevant wholesaler, then aggregates the responses. The retail shop needs to replicate and map some data in this approach as well:

1. The list of categories of each wholesaler, with the mapping between internal category Ids and the retail shop's own.
2. In case the property sets of each wholesaler are different, mappings between the common properties.

The usual situation will be that for each category, there is a core property set, perhaps from a product classification standard, which is supported by all wholesalers, and some wholesalers may offer additional properties.

The virtualized functions are then (changes highlighted in bold):

```
CatalogueItem[] getItems(string categoryInternalId);
Availability getAvailability(sourceId, itemId);
Currency getPrice(sourceId, itemId);
```

The `CatalogueItem` will have to be extended to include the Source Id (identifying the wholesaler) for each item. An alternative would be to construct the retail shop's itemId in a way which allows the extraction of the wholesaler identity and the itemId used by the wholesaler.

The implementation of `getItems` will then look like this (in Pseudo Code, based on Java Syntax):

```
List<CatalogueItem> getItems(String categoryInternalId)
{
      ArrayList<CatalogueItem> result = new ArrayList<CatalogueItem>;
      ArrayList<CatalogueItem> si;
      // for each source in our list of sources
      for(Source src: sources)
      {
            // lookup in our internal mapping table
            String catExternalId =
                  src.lookupCategoryId(categoryInternalId);
            if (catExternalId != null)
            {
                  // get items from source's service
                  si = src.service.getItems(catExternalId);
                  result.addAll(si); // aggregate
            }
      }

      return(result);
}
```

For `getAvailability` and `getPrice`, the retail shop simply routes the request to the wholesaler identified by the Source Id.

In the replicated approach, all catalogues from the wholesalers are periodically replicated into the retail shop's database. All requests from the retail shop's clients can be answered from this database. In addition, availability and price can be requested directly whenever this information is needed up to the minute.

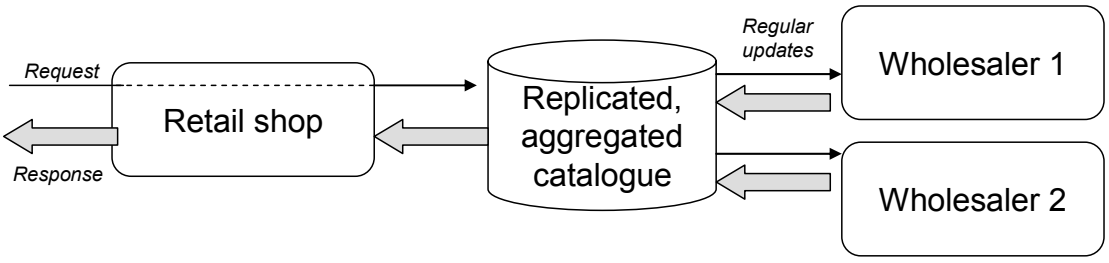


Figure 7: Replicated approach to shop aggregation

This architecture has a number of advantages over the virtualized approach:

- *Reliability:* In the virtualized approach, whenever a wholesaler service goes offline for some reason, its catalogue disappears from the aggregated catalogue. This is hardly expected by users.
- *Performance:* In the virtualized approach, the response takes as long as all individual responses together. This is exacerbated by the current SDL interfaces being synchronous interfaces (see "suggestions" below).

- *Flexibility:* A replicated approach allows the retail shop to add its own information to the individual items in the catalogue. For example, the price it offers to its clients, or sales statistics.

The replicated approach is therefore used by most implementations of this sort today.

The non-trivial part of this approach lies in designing the database to accomodate the property sets, specific to each category, to each wholesaler, and often changing with new versions of catalogues. Using a table for each category can lead to hundreds of tables in the database, and makes searches across categories difficult.

2.2 Simple hotel example

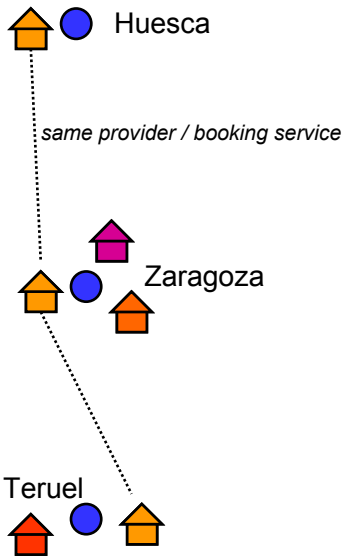


Figure 8: Hotel example illustration

Catalogues are typical for shops selling items, but for services the same catalogue function is necessary, even if consumers do not commonly call it that. As a simple example, consider hotel booking services, each offering a number of hotels. (Each colour in the above figure represents one service.) The items in the catalogue in this case are the different room types offered by the hotel. This is quite stable information, and can therefore be advertised in the BML manifest, as follows:

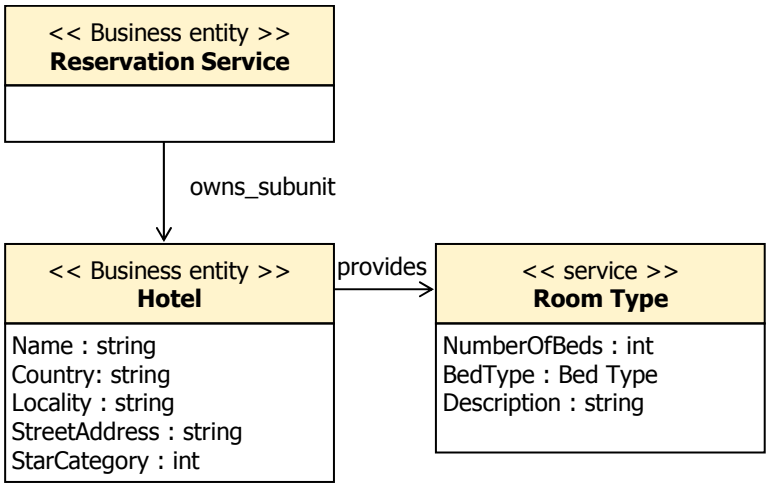


Figure 9: BML model for hotel reservation service

It may appear strange that Reservation Service is a *business entity*, while Room Type is a *service* in this model. This is because BML presumes that one business entity can offer many services, but not vice versa; since a hotel is clearly a business entity, the multi-hotel reservation service has to appear as another business entity. We also should not confuse real-world *business services* (such as "renting hotel rooms") with *software services* (such as "providing a reservation interface"). The BML service metaclass plays the same role as the product metaclass above, it therefore represents a real-world business service.

This BML model enables us to use the built-in Semantic Registry Search functionality to find hotels in a certain location, in a certain star category, etc.

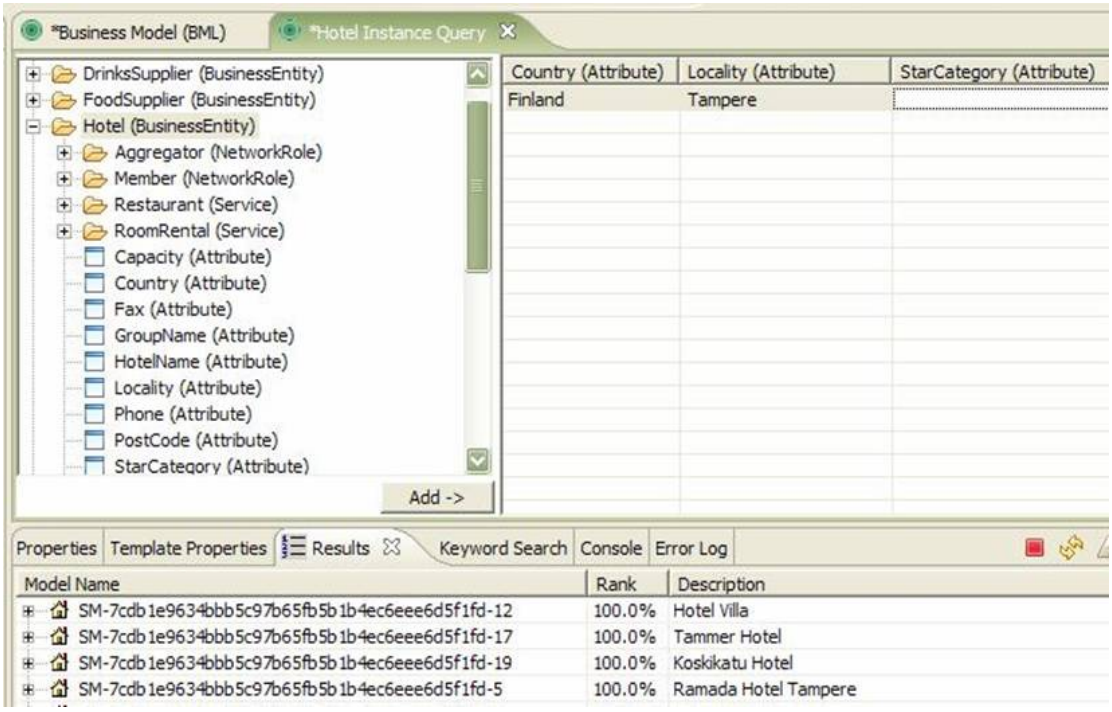


Figure 10: Template search interface using Hotel BML model (prototype)

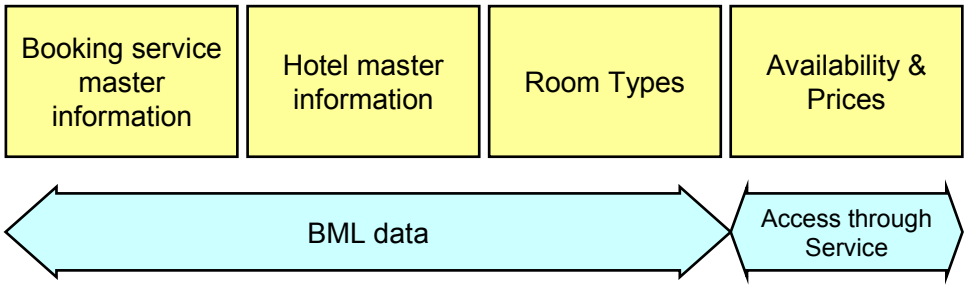


Figure 11: Use of BML for hotel room advertisement

The only information not contained here is availability and price. These are accessed through the service:

```
class Quote
{
      boolean available;
      Currency price;
}
interface ReservationService
{
      …
      Quote GetQuote(String hotelId, String roomTypeId,
            Date arrival, Date departure);
      …
}
```

Here, availability and price are grouped together into a `Quote` object, a very common and extensible approach.

3 Transactions

3.1 Booking a hotel room (introductory example)

A naïve version of a booking interface could look like this:

```
interface ReservationService
{
      …
      // return empty string if unsuccessful
      String MakeRoomReservation(
            String hotelId, String roomTypeId,
            Date arrival, Date departure,
            String firstName, String lastName
      );

      // return true if cancelled
      boolean CancelRoomReservation(String reservationCode);
      …
}
```

The parameters of the `MakeRoomReservation` method mirror those in `GetQuote`. We only need to add some information, e.g. about the person making the booking, special wishes (not shown here), etc.

However, there is a significant problem with this version: What happens if the service or the connection breaks down after the request has been sent, but before the response has arrived? The client then has no way of knowing if the reservation has actually been made. If the server did in fact send the response, it has no way of knowing if the client received it.

Another problem could occur in the following scenario: The client has requested a quote at 4:00 PM. At 4:05 PM the price of the room type was raised, perhaps because only few rooms are left. At 4:10 PM the client sends the reservation, assuming that it's still the price in the quote.

3.2 Making orders durable

In order to make a business transaction durable over an insecure connection, several exchanges are necessary. The simplest way to fix this problem is to make sure the client has received the reservation code before committing the order. This way the client can always synchronize its view of the world with the server's, once the connection is re-established. New elements are highlighted in bold:

```
class ReservationResponse
{
     boolean    success;
     String     reservationCode;
     String     hotelId;
     String     roomTypeId;
     Currency   price;
     String     message;
}
interface ReservationService
{
     …
     // room reservation will be held for 10 mins
     ReservationResponse CreateRoomReservation(
          String hotelId, String roomTypeId,
          Date arrival, Date departure,
          String firstName, String lastName
     );

     // return state: 0:committed, 1:uncommitted, 2:not found
     int GetReservationState(String reservationCode);

     // return true if successful
     boolean CommitReservation(String reservationCode);
     boolean CancelRoomReservation(String reservationCode);
     …
```

```
}
```

After `CreateRoomReservation`, the room in question is temporarily blocked in the backend reservation system. If, within 10 minutes, the client does not send a `CommitReservation` request, the reservation will be cancelled.

The `ReservationResponse` contains all the essential information about the reservation, and is therefore equivalent to a legal offer.[6] There is also room for an additional message, e.g. reason for an unsuccessful request, or relevant remarks regarding the successful reservation. The client should store this information to a non-volatile medium before proceeding; the server should save it anyway. The `CommitReservation` request is legally equivalent to the acceptance of the offer.

The connection between client and server can break down at any time during this exchange. The reservation on the server can be in three states: a) the reservation was successfully committed; b) the reservation is in the temporary, uncommitted state; c) the reservation does not exist, either because it has never been created, cancelled, or expired from the uncommitted state. Using `GetReservationState`, the client can find out about this state.

3.3 Computing equipment example

In the case of computing equipment, the major difference lies in the fact that customers normally want to buy several items in one order. One payment and one shipping is tied to the order.

Theoretically, it would be possible to implement a sort of *shopping cart* interface as the SDL interface, where customers can create a shopping cart, add items to it, then go to the checkout. This would mimic the HTML interface offered today by e-commerce sites. However, this is not appropriate for automated B2B interaction. The server (shop) does not need to know anything about the order before it is actually sent. A lot of unnecessary method calls would be exchanged. In order to provide a responsive user interface, the client would have to cache the information anyway, and so on.

In order to work with a coarse-grained approach appropriate for B2B, we therefore need a few more complex value classes to represent the order information:

```
class OrderItem
{
     String   itemId;
     int      quantity;
     Currency price; // not set by client
```

[6] (From a security point of view, it would be appropriate to send an electronic signature with this offer, so that it becomes undisputable. However, since the infrastructure for this is not in place yet, we do not describe that aspect here.

```
}
class ShippingInfo
{
     String    shippingAddress;
     String    shippingServiceId;
     Currency  shippingFee; // not set by client
}
class PaymentInfo
{
     String    paymentProviderId;
     String    customerId;
     String    customerName;
     String    credentials;
     Currency  paymentFee; // not set by client
}
class Order
{
     OrderItem[]    items;
     ShippingInfo   shippingInfo;
     PaymentInfo    paymentInfo;
     Date           orderedDate; // not set by client
     Date           shippedDate; // not set by client
     Currency       totalAmount; // not set by client
     String         orderId; // not set by client
     int            state; // not set by client
}
```

The information about available shipping and payment services could come from the BML manifest of the shop:

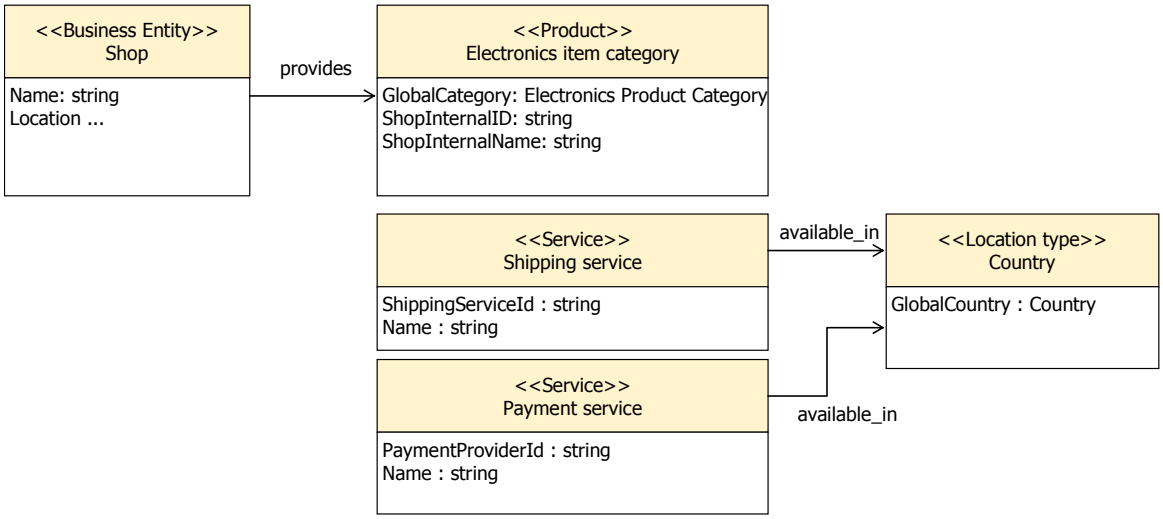


Figure 12: Extended BML model for shop's shipping and payment services

The set of all countries can be managed as a concept in the ontology and used to describe which shipping and payment services are available where.

An object of the class `Order` can then be constructed by the client. Each order item consists of an item ID (from the catalogue) and a quantity. The price does not have to be set by the client. Likewise, `PaymentInfo` and `ShippingInfo` are filled.

The interface then follows the same pattern as above:

```
interface EquipmentOrderingService
{
      …
      // order will be held for 20 mins
      Order CreateOrder(Order myOrder);

      // return order including current state,
      // null if not found
      Order GetOrder (String orderId);

      // return true if successful
      boolean CommitOrder(String orderId);
      boolean CancelOrder(String orderId);
      …
}
```

As the response to `CreateOrder`, the server will send an `Order` object back; in the case of an error, it will give back a failure code in the `state` attribute. In the case of success, it will send back the full order description, including the data necessary to make it a complete offer in a legal sense: Prices for each item and for the added services, as well as an order Id under which the order can subsequently be identified. The client should save this information before committing the order. In case of a failure, the client can always find out about the current state of the order through `GetOrder`.

3.4 Customer accounts

Another possibility to recover in the case of a connection failure is to create a customer account, linked to a DBE identity. Once the client has opened a session under its identity (the precise API is currently finalized), it can retrieve the list of Order Ids using

```
String[] GetMyOrderIds();
```

and access each Order using GetOrder.

3.5 The role of e-mail

If you think about it, most current e-commerce sites cannot function without e-mail. E-mail has the advantage of being asynchronous, hence does not depend on the stability of connections, ubiquitous (everyone has it) and it can usually be expected to be read within a few days. E-mail addresses are verified during the creation of customer accounts. E-mail is used to recover in case users lose

their password. All transactions are usually documented by an associated e-mail. Customers can be notified of any problem that emerges while their order is being processed, without them having to log on periodically and check, which would be very inconvenient.

Of course, e-mail spam has become such a big problem now that a service provider cannot really rely on e-mail being read. But still, when it has send two notifications concerning any serious problem, the service provider is said to have fulfilled its duty.

Perhaps the integration of e-mail, along with some features addressing the security and spam problems, should be part of a future architecture for transactions.

3.6 Distributed transactions

In the tourism and conference organizing scenarios discussed for the Aragon and Tampere regions, there are many occasions when an organizer has to book several resources for one event: For example, a flight, a hotel and a catering service. If any of these resources cannot be reserved, the often the whole set should not be.

Although this resembles a distributed transaction discussed in database research, it is probably not a good idea to treat it on the database level; and in fact, current e-commerce solutions do not expose the database transaction interface to the clients. In databases, the goal is to be able to roll back the database, in case of a failed transaction, to the very state it was in before committing it. The longer the transactions run, and the more transactions are performed, the more resources this costs on the database server, and the more it raises the risk of resources becoming blocked.

For real-world business transactions, there is no need to roll back to the previous state. All we need is a possibility to cancel an operation, at least a few minutes after it has been committed.

We can then implement the distributed transaction using the Create / Confirm / Cancel methods introduced above.

The client acts as the "transaction coordinator". First, it asks all service providers to create the order. All service providers should then temporarily reserve the necessary resources (hotel rooms, flight seats, etc.), so that under normal circumstances, and if a Confirm request is sent within a few minutes, their part of the transaction will succeed. If the client receives an "unsuccessful" response or no response from any of the service providers, it cancels the transaction by either sending a `CancelOrder` request, or simply waiting for the orders to expire.

If all service providers have sent a "successful" response, the client then sends a Confirm request to all. Since usually little time passes between the two events, it is quite unlikely that anything will go wrong now. But if it does, if for example a connection breaks down, the client can do two things: 1. It can resend the Confirm request to services that have not responded, recreating the order if necessary. 2. If that is unsuccessful, it can cancel the orders of all the other services.

For this, it would be useful to define a minimum cancellation period, for example 30 minutes (except for orders which are fulfilled on the spot, such as buying online music).

4 Collaboration

For the conference organizing scenario, it would be very helpful if all parties involved in the organization had a common view of the events of the conference week, so that potential conflicts are discovered early on. As a simple example, we will therefore discuss a project calendar service, and its integration with each participant. Similar architectures can be used for managing any kind of object in a common workspace.

4.1 Project calendar service

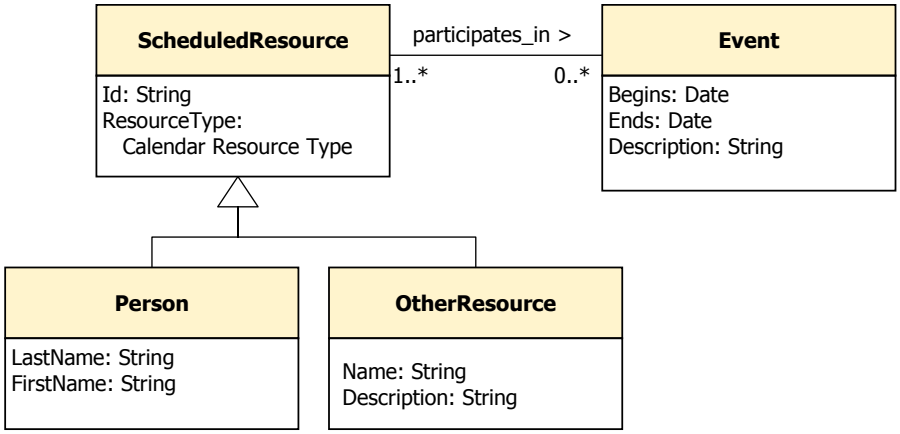
As with catalogues, we have the choice between a completely virtualized approach and a replicated approach. In the virtualized approach, each party stores its part of the data, and requests are answered by routing it to all parties involved. As before, a replicated approach is much more robust, since it does not require all participants to be online. We have the choice between centralized replication and Peer-to-Peer replication. Peer-to-Peer replication is more robust against failure, but is much more complex to build. The Peer-to-Peer infrastructure in DBE is built for the low-level exchange of service proxies with unique identifiers, not for arbitrary business objects which are queried based on their properties. It also does not work well (as any Peer-to-Peer architecture) for data which is frequently updated. Therefore we will use a centralized replication architecture.

There is therefore a project service running on one node. The operations it will support are:

- Managing master data of people and other resources involved in a scheduled event
- Querying the events a person or resource has in a given time span

- Scheduling an event; deleting an event.

The conceptual model behind the interface is like this:



The interface could look like this:

```
Person[] GetAllPersons();
Person GetPerson(String resourceId);
OtherResource[] GetAllOtherResources();
OtherResource GetOtherResource(String resourceId);
Event[] GetEvents(String resourceId, Date from, Date until);
void ScheduleEvent(Event e, String[] participantIds);
```

A typical calendar client application like Mozilla Sunbird could be extended to access this interface. [7]

4.2 Group calendar as calendar aggregation

In collaborative projects like conference organization, the organizing staff does not only work on one project, or with one group of companies. Therefore, although a project calendar is useful for scheduling the conference (the time spent by people who attend the conference), it may not in itself be very useful for the organizing team. Each member of the team usually wants to have an integrated view of all scheduled events.

This aggregation can be achieved using the following architecture; it contains both elements of the virtualized approach and the replicated approach introduced in section 2.1.5. The individual organizations calendars are replicated in a centralized database to allow Read access to the calendar even when they are

[7] At the moment, a protocol based on WebDAV methods and the iCalendar file format is supported by many calendar clients. Our example does not introduce any new "killer features" which would really justify creating another interface; it only serves as an instructive, well-known example for the general category of collaborative applications. The WebDAV-based protocol supports very similar operations to those described above, which validates the approach described.

offline (`GetEvents`). When the project calendar receives a GetEvents request, it looks up the calendar responsible for the person or other resource. If the calendar is online, it forwards the GetEvents request to the calendar. The response of this request can the be used to update the backup. If the calendar is offline, it retrieves the events from the backup database. Requests to read the master data of people and other resources can be treated in the same way.

On the other hand, schedule requests require the responsible calendar to be online. If the responsible calendar is offline, the schedule request returns with an error.

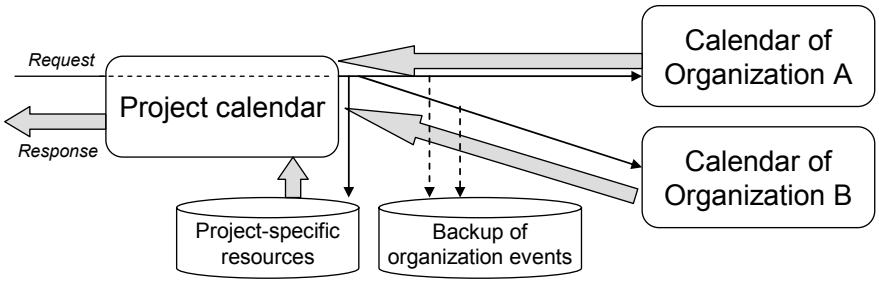


Figure 13: Flexible architecture for aggregated project calendar

Calendars should be able to register themselves with the project calendar (an administrator can then accept or reject the registration request; or perform additional registrations without the organization calendars explicitly requesting it). The calendars should also notify the project calendar when they come back online after being offline for a while. So the project calendar needs these additional operations:

```
void RegisterCalendarResources(Node node, String[] resour-
ceIds)
void NotifyConnect(Node node)
```

The parameter `node` identifies the endpoint of the service (Fada node or IP address).

5 Conclusion and Suggestions

The examples developed in this document have shown that it is possible in principle to implement the diverse applications in typical B2B scenarios. However, the DBE infrastructure, in its current state, does not always offer the critical building block necessary to implement them – many more are necessary.

In devising these examples, we could identify a few rather easy extensions to the current infrastructure which, if made, would greatly help in the implementation of typical scenarios. They are described in the remaining sections:

5.1 Using Ontologies / BML in SDL methods, outside of the Service Registry

There were a number of arguments, such as reliability, in favour of the current Knowledge Base and Semantic Service Registry implementation as a distributed Peer-to-Peer database. However, our discussion of catalogues has shown that this approach also limits the applicability of the Registry for any data which are frequently updated or large in volume. These technical critieria create a deep division in the infrastructure, where conceptually there should not be one. For example, the property sets which describe a product category are really like a model of that category, and we should therefore be able to use the BML editor and the BML language to read and write these models, even if they are highly dynamic, or should only be visible to established business contacts, and therefore cannnot be published in the Semantic Service Registry.

One way this could be achieved is by providing the BML framework as a standalone technology, with a set of tools (e.g. editors), a standardized XML serialization of BML data, models and the metamodel, and a query language. For various programming languages and software service infrastructures, support can the be added for using BML classes as data types, accessing BML data and issuing queries. This technological aspect could be completely separated from the challenge of providing a central, searchable and highly reliable repository.

5.2 Proposal for separating concerns in BML

5.2.1 Background

The current idea of BML in DBE more or less assumes that BML models (M1 level, sentence types or whatever the equivalent in SBVR) will be tailored to individual sectors so as to be able to capture the specific semantics of each one.

While this is good in terms of expressive power and flexibility, it may create a problem of costs involved in order to achieve interoperability between businesses from different sectors. We should expect business partners to come from different sectors as the rule rather than the exception, since there is usually no point dealing with your direct competitors.

5.2.2 Separating Interaction Patterns from Product Descriptions

I therefore suggest a way to reduce complexity and enhance reusability and interoperability by separating two concerns, or aspects: The first aspect is the product or service description schema which almost certainly will be sector specific, even product type specific. A hotel room has different attributes than a flight, a taxi ride or a piece of downloadable music. The second aspect is the interaction pattern, or interaction process between provider and customer of the product or service. Here, hotel, flight and taxi ride have a lot in common: They all are services provided for a certain time span; they can be reserved ahead of

time; and it may or may not be possible to cancel this reservation. Their price varies with both the time for which they are reserved (time of service provision) and with the time of reservation, usually dependent on the amount of capacity still available. The piece of downloadable music, on the other hand, is treated according to a very different interaction pattern. There is no such thing as a reservation; it is more likely that we find quite complex pricing schemes, with monthly subscription fees, special discounts for a limited time etc. But again, these interaction patterns could be found also for other products in other sectors, such as image agencies (for use in advertising), premium weather forecasts, etc.

5.2.3 Benefits

The first benefit of this separation is obviously a reduced effort of creating the necessary BML models in the first place. But there is also a "run-time" benefit of enhanced interoperability:

A given business (or: its business software) cannot be expected to understand and deal with the evolving BML models of the hundreds of sectors its suppliers come from.

But with the separation mentioned, this is not necessary. A useful, universally known concept in electronic business is the ABC classification of suppliers. It comes from industry but can equally be applied to services. A suppliers are strategic suppliers which supply only few types of goods, but those with high value and impact. A typical example would be Bosch as a supplier of diesel injection systems for an automobile manufacturer. C suppliers supply a vast array of commodity goods, easily exchangeable, with limited value; for example, office supplies. B suppliers are in the middle between the two. Now, a business will seek tight integration of it's production and logistics systems mainly with A suppliers. Only for those A goods does the business' IT systems need to look deeply into the goods' characteristics, will rules be defined as to which supplied good A1 can be combined with which supplied good A2 for a given product or service, where joint forecasting will be done, etc. C supplies, on the other hand, will appear as quite anonymous "items" where only the cost is relevant and will be processed.

On the other hand, even C supplies may be carefully selected manually by the customer company's employees. All the power of BML for product description is still useful to them in order to identify the products or services that best fit their requirements, but the BML data does not have to be processed by their own IT systems. Take a laser printer for normal office use, for example. The employee selecting the laser printer will look in depth at print speeds, maintenance costs, etc. which can be described using the BML product module. But once the laser printer is ordered, it will appear to the customer's IT system as an anonymous "office item" with an expenditure, a cost centre paying for it, etc.

In other words, the customer company may choose to only integrate its IT system with the interaction process defined by the supplier, benefiting from automated billing etc., but not with its product/service description schema.

5.2.4 Consequences for BML

What is needed therefore is a careful packaging of BML models, as well as inheritance mechanisms between packages. The BML metamodel already provides a useful division of the metamodel into several modules (product, organisation etc.). The BML model of a sector, or a company (if it wants to use a specific one) will be a composition of one or several product/service description packages, with a chain of inheritance leading eventually up to the generic product/service description of the generic package, and other packages such as the process package or organisation package, each with their own inheritance chain.

5.3 Support for coarse-grained, asynchronous data exchange and replication

Whenever there is a need to create aggregated services, as shown here for catalogues and project calendars, there is very likely the need to replicate certain data in order to maintain the aggregated service even if not all the constituent services are available. When performing transactions which have legal implications (such as an order, or an invoice), all the legally relevant information should be sent together, so it can be electronically signed, timestamped, and archived. The earlier EDI standards, as well as many recent B2B initiatives used by large corporations (e.g. Oasis) therefore concentrate on the definition of document formats. E-mail was mentioned as an important element in the currently dominating e-business experience of customers. The current SDL infrastructure, as any Web services framework, often favours an operation-oriented, fine-grained, and synchronous approach – which is also needed by many applications. But the "bread and butter" processes in e-business would be better served by a document- and data-centered infrastructure, with coarse-grained operations, asynchronous data exchange (such as secure e-mail channels), and replication.