Configuring Business Objects from Legacy Systems

Willem-Jan van den Heuvel, Mike Papazoglou, and Manfred A. Jeusfeld

INFOLAB, Tilburg University, PO Box 90153,
Tilburg 5000 LE, The Netherlands
wjheuvel@kub.nl, mikep@kub.nl, jeusfeld@kub.nl

Abstract. In this paper we present a methodology, called binding Business Applications to LEgacy systems (*BALES*), that allows to blend modern business objects and processes with objectified legacy data and functionality in order to construct flexible, configurable applications, that are able to respond to business changes in a pro-active way.

1 Introduction

As a result of the growing turbulence in which modern organizations operate, the design of information systems is faced with new challenges. The adaption and deployment of information systems needs to be completed in the shortest possible time, amidst changes, as the organizations and their constituting business processes tend to become more complex every day [1]. Accordingly, most organizations are striving to respond to rapid changes by creating modular business processes that can be quickly implemented and re-engineered as the situation may demand [2].

To meet the requirements of modern organizations, and get better reuse from software, distributed business object computing is the preferred solution [3]. Business objects can be the key building block in the re-engineered (process-oriented) enterprise as they can realize domain business processes and default business logic that can be used to start building applications in these domains. Furthermore, domain specific models can be designed as business frameworks so that they can be easily extended and modified, e.g., SAP and IBM's San Francisco business objects [4]. These can be deployed for integrated enterprise-wide applications that can be easily built upon distributed broker architectures such as CORBA. However, most contemporary enterprise information systems are characterized by a rigid (technical) infrastructure and their heritage of data to perform their primary processes. These systems are not able to keep abreast of the rapid organizational and technological changes that occur in a business environment. Such information systems 'that significantly resist modification and evolution to meet new and constantly changing business requirements' can be defined as *legacy systems* [5].

Over the years several strategies to deal with the legacy problem have been proposed: access integration in place, the cold turkey (replace at once) and the gradual migration approach [6], [5]. Access/integration in place requires an environment in which the legacy systems and new business components can coexist and cooperate. For that purpose it uses technologies such as object wrappers [7]. Wrappers are used to objectify legacy systems and expose interfaces over legacy transactions as well as provide meta-data descriptions of legacy data [8]. Such wrapping solutions generally present the following four drawbacks:

M. Jarke, A. Oberweis (Eds.): CAiSE'99, LNCS 1626, pp. 41–56, 1999.

1. The object wrapping solution views enterprises from the legacy perspective and assumes that new applications may be developed in terms of legacy objects with perhaps marginal adjustments to the legacy data and functionality. This assumption is not realistic given that the legacy systems are geriatric systems that reflect organizational requirements and objectives of a long time ago.
2. Because of the way that legacy applications are built and continue to be upgraded, they do not only contain data divorced from a business context but they are also seldom consistent with modern business objectives.
3. It is unlikely that this approach would survive in an increasingly fluid environment where (unpredictable) organizational changes and business objectives require frequent adjustment and enhancement of the legacy functionality.
4. Finally, and possibly more importantly, any viable long-term approach to mapping between legacy and target systems must be *methodology-oriented* and be performed on an ad hoc basis as the situation may demand.

In this paper we propose a methodology, called binding Business-Applications to LEgacy Systems (*BALES*). The *BALES* methodology allows to blend modern business objects and processes with legacy objects and processes to construct flexible applications. This methodology allows reusing as much of the legacy data and functionality needed for the development of applications that meet modern organization requirements and policies. This implies ”adjusting” (or retrofitting) legacy data and functionality at the enterprise modeling level. In particular, the *BALES* methodology allows to construct configurable business applications on the basis of business objects and processes that can be parameterized by their legacy counterparts. The goal of the *BALES* methodology is to enable organizations to react at business induced changes in a manner that does not disrupt enterprise applications or the business processes that underly them.

The remainder of this paper is organized as follows. In the next section, we present the *BALES* methodology for linking business objects and processes to objectified legacy data and functionality. In section 3, we present a realistic example to illustrate the application of the *BALES* methodology. Finally, section 4 describes our conclusions and future research directions.

2 A Methodology for Binding Business Application Objects and Processes to Legacy Systems

Most of the approaches to integrate legacy systems with modern applications are designed around the philosophy that data residing in a variety of legacy database systems and applications represents a collection of entities that describe various parts of an enterprise. Moreover, they assume that by combining these entities in a coherent manner with legacy functionality and objectifying (wrapping) them legacy systems can be readily used in place. In this way it is expected that the complexities surrounding the modern usage of legacy data and applications can be effectively reduced. Unfortunately, these approaches do not take into account the evolutionary nature of business and the continual changes of business processes and policies. Although part of the functionality of a legacy system can be readily used, many of its business processes and policies may have changed with the passage of time.

A critical challenge to building robust business applications is to be able to identify the reusable and modifiable portions (functionality and data) of a legacy system and combine these with modern business objects in a piecemeal and consistent manner. These ideas point towards a methodology that facilitates *pro-active change management*

of business objects that can easily be retrofitted to accommodate selective functionality from legacy information systems. In the following we describe such a methodology that takes into account these considerations. This methodology concentrates on parameterizing business objects with legacy data and functionality. However, the same methodology can be successfully employed for coping with changes to existing business objects and processes.

One important characteristic of business object technology, that also contributes to the critical challenge described above, is the explicit separation of interface and implementation of a class. Business objects technology takes this concept a step further by supporting *interface evolution* in a way that allows the interfaces of classes to evolve without necessarily affecting the clients of the modified class. This is enabled by minimizing the coupling between business components. Client and server classes are not explicitly bound to each other, rather messages are trapped at run-time by a semantic data object that enforces the binding at the level of parameter passing semantics [13]. As we will see in the following, the *BALES* methodology thrives on this key feature of business object technology.

2.1 The *BALES* Methodology

The *BALES* methodology, that is under development, has as its main objective to parameterize business objects with legacy objects (LOs). Legacy objects serve as conceptual repositories of extracted (wrapped) legacy data and functionality. These objects, just like business objects, are described by means of their interfaces rather then their implementation. A business object interface can be constructed from a legacy object interface partition comprising a set of selected attribute and method signatures. All remaining interface declarations are masked off from the business object interface specification. In this way, business objects in the *BALES* methodology are configured so that part of their specification is supplied by data and services found in legacy objects. A business object can thus have a part that is directly supplied from some legacy data and services which it combines with data and services defined at its own level. This means that business object interfaces are parameterizable to allow these objects to evolve by accommodating upgrades or adjustments in their structure and behavior.

The *BALES* methodology borrows ideas from the object-oriented application development literature based on *use cases* [9] and *task scripts* [14]. It also combines ideas from event-driven business process (re-)engineering [15], with concepts from the area of enterprise modeling [11], [16]. *BALES* presents some similarities with contemporary approaches in the field of Enterprise Resource Planning (ERP) package development, e.g., the San Francisco-project of IBM [4]. Lastly, *BALES* draws on recent research to workflows and interoperability, e.g. [30].

The core of the *BALES*-methodology comprises the three phases (see fig. 1): *forward engineering*, *reverse engineering* and *meta-model linking*. To illustrate the *BALES* mapping methodology a simplified example is drawn from the domain of maintenance and overhaul of aircrafts (see fig. 1). This example was inspired from building block definitions that we currently help develop at the Department of Defense in the Netherlands [17]. The upper part of this figure illustrates the results of the forward engineering of the business domain (phase 1) in terms of workflows, business processes and business objects. As can be seen from this figure the enterprise model is enacted by a *Request-Part* workflow which comprises three business processes: *Request*, *Prognosis* and *Issue*. The *Request-Part* workflow is initiated by a maintenance engineer who requests parts (for maintaining aircrafts) from a warehouse. A warehouse manager can react in two different ways to such a request.

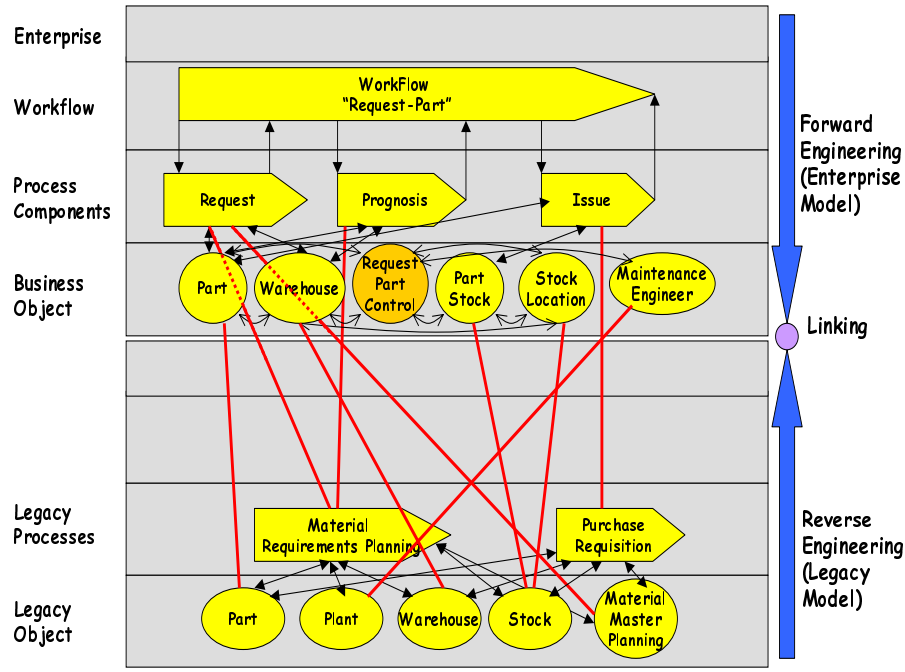

Fig. 1. Developing an enterprise model by means of reusing legacy processes and objects.

Firstly, the manager can directly issue an invoice and charge/dispatch the requested products to the requester. In this case, the workflow will use information from the *Request* process to register the maintenance engineer's request in an order list. This list can be used to check availability and plan dispatch of a specific aircraft part from the warehouse. The *Request* process uses the business (entity) objects *Part* and *Warehouse* for this purpose. Subsequently, the workflow initiates the *Issue* process (see fig. 1). The *Issue* process registers administrative results regarding the dispatching of requested part and updates the part inventory record by means of the *Part-Stock* business object. The business object *Request_Part_Control* is an auxiliary control object used during the execution of the workflow to store and control the state of the running business processes. If the requested part is not in stock then an *Order-Part* workflow is triggered (not shown in this figure). This workflow then orders the requested parts to fulfill the request of the *Request Part* workflow.

Secondly, in case of an 'abnormal' request, for example if the customer informs the warehouse manager about a large future purchase, the manager may decide to run a prognosis. This process scenario first registers the request information provided by the *Request* business process and runs a prognosis on the basis of the availability and consumption history of the requested part. The *Prognosis* process uses information from the *Part* and *Warehouse* business objects for this purpose. After the prognosis has ran successfully the part can be reserved. If the results of the process *Prognosis* are negative, as regards future availability of the requested aircraft part, the workflow *Order-Part* is activated.

The lower part of the picture 1, represents the result of the reverse engineering activity in the form of two processes (wrapped applications and related database(s)) *Material_Requirements_Planning* and *Purchase_Requisition*. These processes make use of five legacy objects to perform their operations. Moreover, figure 1 indicates that the enterprise workflow draws not only on "modern" business objects and processes, but also already existing (legacy) data and functionality to accomplish its objectives. For example, business processes such as *Request* and *Issue* on the enterprise model level are linked to the legacy processes *Material_ Requirements_Planning* and *Purchase_Requisition* as indicated by means of the solid lines. This signifies the fact that the processes on the business level *reuse* the functionality of the processes at the legacy model level. The same applies for business objects at the enterprise model level such as *Part*, *Part-Stock* and *Stock-Location* which are parameterized with legacy objects. In this simplified example we assume that problems such as conflicting naming conventions and semantic mismatches between the enterprise and legacy models have been resolved. A solution to this problem can be found in [18].

Figure 2 represents the individual steps and (intermediate) milestones during the three main phases of the *BALES* methodology. These are described in the following subsections.

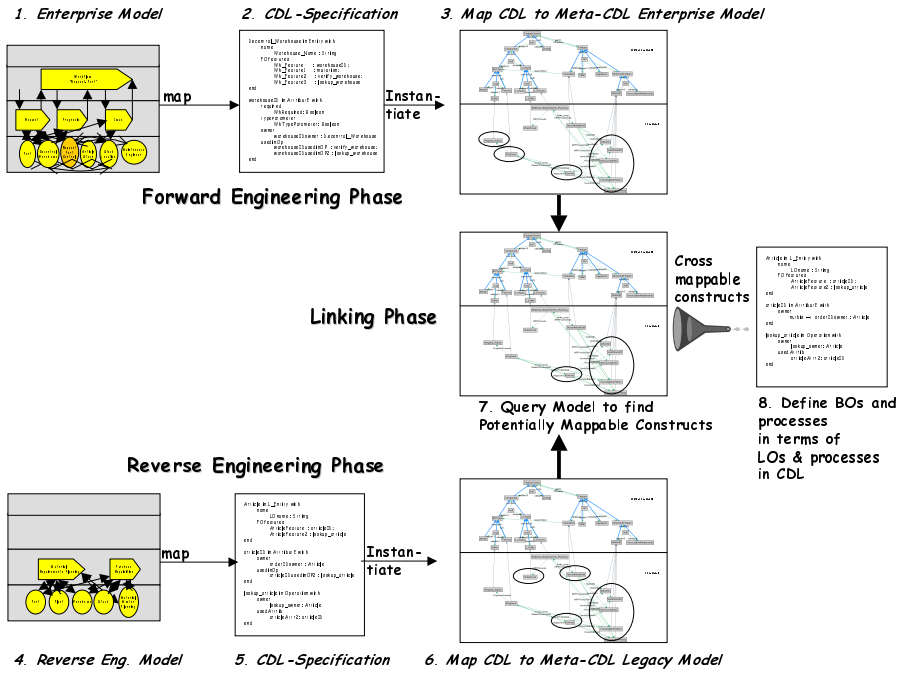

Fig. 2. The *BALES* methodology.

2.2 Forward Engineering the Business

The forward engineering phase transforms a conceptual enterprise model into CDL and maps this CDL definition to a Meta-CDL-Model which serves as a basis for comparison

between business and legacy enterprise models. This phase comprises the following activities which correspond to steps 1, 2, and 3 in fig. 2.

1. *Enterprise Modeling:*
   The forward engineering activity starts with the construction of an enterprise model. The enterprise model reveals the activities, structure, information, actors, goals and constraints of the business in terms of business objects, processes and workflows (that can be defined in terms of each-other), and is typified by the enterprise workflow in the upper part of fig. 1.
   As can be seen in this figure, the enterprise model is structured by means of a layered enterprise architecture. This architecture comprises four layers: the (atomic) business objects, business processes, business workflows and business policies/goals. Business objects provide a natural way for describing application-independent concepts such as product, order, fiscal calendar, customer, payment and the like. A business process is used to define a set of interrelated activities that collectively accomplish a specific business objective, possibly, according to a a set of pre-specified policies. The purpose of this layer is to provide *generic business processes* in terms of business object services. The workflow layer assigns activities to actors according to the state of each process in progress and moves the process forward from one activity to the next. Lastly, the policy layer constitutes the business policies in terms of subsequently the workflow, process and/or business objects. An elaborated description of the enterprise framework can be found in [12].
2. *CDL-Specification of the Enterprise Model:*
   The interface descriptions of the business objects and processes need to be constructed on the basis of the enterprise model.
   To formally describe the interfaces of business objects we use a variant of CDL that has been developed by the OMG [19]. CDL is declarative specification language – a superset of OMG IDL, ODGM Object Definition Language (ODL) and the ODGM Object Query Language – that is used to describe composite behavior of communities of related business objects. A specification in CDL defines business object interfaces, structural relationships between business objects, collective behavior of related business objects and temporal dependencies among them [19]. An object defined using CDL can be implemented using any programming language as long as there exists a CDL mapping for that language, e.g., Java and Smalltalk. Practical experiences with use of the CDL can be found in [20].
3. *Instantiating the Meta-CDL Enterprise Model:*
   After the interfaces of both the business objects and processes have been specified in CDL, the CDL specifications are instantiated to a Meta-CDL Enterprise (Business) Model. This model depicts the *instantiations* of the CDL enterprise model components. It thus illustrates how the CDL and model specific constructs are related to each other, and provides information about their types. The CDL meta-modeling step is used as basis to infer how the constructs found in a Meta-CDL Enterprise Model can be connected to (viz. re-use) related constructs found on the Legacy Model (see section 2.4). In summary, the Meta-CDL-Model serves as an 'independent' canonical model to which the forward as well as the reverse engineered CDL models will be linked, superimposed, and compared in order to ascertain which (portions of) legacy processes and objects can be reused at the enterprise model level. In this way, it is possible to parameterize enterprise model business processes and objects with related legacy business processes and objects.

2.3 Reverse Engineering the Legacy System

In the second phase of the *BALES*-methodology, we represent the legacy objects and processes in terms of CDL and link them to a Meta-CDL Legacy Model. The activities during the reverse engineering phase are similar to those performed during the forward engineering phase. The following activities, which correspond to steps 4, 5 and 6 in fig. 2, can be identified:

1. *Reverse Engineered Model:*
   The reverse engineered model represents the wrapped legacy data and functionality. To construct the legacy objects we rely on techniques that combine object wrapping and meta-modeling with semantic schema enrichment [23], [24].
   The legacy object model comprises a distinct legacy object and legacy process layer in the Enterprise Framework (see bottom part of fig. 1).
2. *CDL-Specification of the Legacy Model:*
   The interfaces of the legacy objects and processes are described by CDL in the same way as we explained for business processes and objects.
3. *Instantiating the Meta-CDL Legacy Model:*
   After the CDL-descriptions of the legacy components are available the legacy CDL specifications are instantiated to a Meta-CDL Legacy model much in the same way that we described for the enterprise model.

2.4 Link Phase of the CDL Meta Models

The CDL descriptions of both the forward- and backward-engineered models have to be connected to each other in order to be able to ascertain which parts of the legacy object interfaces can be re-used with new applications. To achieve this, we represent both business and legacy CDL specifications in a repository system. The repository system has a CDL meta model which is capable of representing the CDL constructs and the CDL upgrades that we introduced for representing legacy object interfaces and their components. The advantage of this repository approach is that the content of the repository, viz. Meta-CDL Models, is subject to automated analysis, mainly by means of queries. For this purpose we utilize the ConceptBase system [22] because it has an advanced query language for abstract models (like the CDL meta model) and it uniformly represents objects at any abstraction level (data objects, model components, modeling notations, etc.).

The underlying representation language of ConceptBase is Telos [21]. Telos has a frame syntax to represent classes and objects. An equivalent representation is in form of directed graphs. The content of a ConceptBase repository is subject to queries. The query language is based on deductive rules. In the frame syntax, queries have the form:

```
QueryClass <query-name> isA <class>
  retrieved_attribute
   <attr-name> : <Class>
   ...
  parameter
   <param-name>: <Class>
   ...
  constraint
    <con-name>: <membership-condition>
end
```

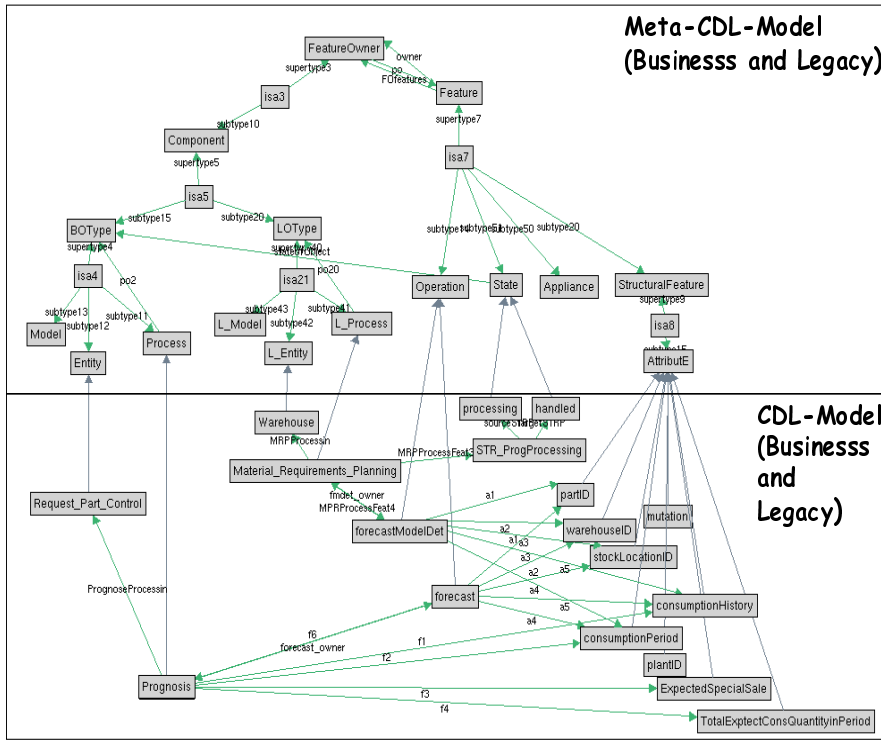

Fig. 3. A snapshot of ConceptBase graph browser containing the Business and Legacy Meta-CDL Models.

The interpretation of a query are all instances of the super class (`isA`) which fulfill the membership condition. The system will include the retrieved attributes in the answer. If parameters are specified, the user can call a query with values for the parameters. Parameters may appear in the membership condition which itself is a logical formula.

Figure 3 shows an excerpt of the abstract representation of business and legacy objects displayed by the ConceptBase graph browser. The upper half shows the Meta-CDL-Model, i.e. the meta classes to represent CDL constructs. Note the distinction between business object types (*BOType*) and legacy object types (*LOType*). The lower half displays an excerpt of the result of the instantiations of CDL representations to Meta-CDL models for business and legacy objects. Note that *Material_Requirements_Planning* is classified as legacy object type whereas *Prognosis* is instantiated as business object (*BOType*).

After the CDL descriptions of the forward and reverse engineered models have been instantiated to their respective Meta-CDL-Models, the two models can be superimposed and compared (see fig. 3). The following two tasks need to be performed in succession in oder to be able to link legacy objects to business objects.

1. Query the Model to find Potentially Mappable Components
   During this task queries are used to infer potential legacy components that may be connected to business components. For instance, we can identify business object

attributes and/or operations that can be constructed out of legacy object attributes and/or operations. In [25] ConceptBase is successfully deployed for solving a similar problem. Telos queries are used to retrieve exact or partial matches of signatures of requested components that are stored in a repository. As argued in [25], most queries will lead to a *partial solution*, since it is unlikely that two interfaces will match exactly. This type of querying is useful for combining signatures of legacy operations with operations of business objects. However, it raises type safety issues that will be addressed in section 4.

2. Specify Business Objects in CDL in terms of Legacy Objects
   The legacy constructs that are returned by the query answers above are subsequently substituted in the CDL business object specifications obtained in step 2 of the forward engineering phase, see fig. 2. For this purpose we have extended the CDL language with a special linking operator 'X $\mapsto$ Y', where X represents some source construct (for example a Meta-CDL construct for business *Operation*, see fig. 3). Y denotes the target construct (e.g., a Meta-CDL legacy *Operation*) which corresponds to a reused operation from the Meta-CDL legacy model.

The *BALES* methodology results in a CDL specification of business objects and processes in terms of their related legacy counterparts. The mapping statements can be easily adjusted to satisfy new market requirements by, for example, accommodating new packages such as ERP solutions [10], [4].

In the next section, we will illustrate the *BALES* activities and milestones in terms of the aircraft maintenance and overhaul example that we presented earlier.

3 Putting Things Together

3.1 Forward Engineering

In the following, we will explain the forward engineering phase in a step-by-step manner according to what we have outlined in section 2.

1. *Enterprise Modeling:*
   During this phase the enterprise model is constructed as already explained in section 2.1.
2. *CDL-Specification of the Enterprise Model:*
   The enterprise model represented in fig. 1, serves as a starting point to specify the business object/process in CDL. We shall give an example of a CDL specification involving a business object with interesting dynamic behavior, namely the *Request_Part_Control* in fig. 2. Detailed descriptions of the CDL syntax can be found in [19].
   This CDL specification describes the interface of the business control object *Request_Part_Control* (see fig. 1) and shows that this business object encapsulates three business processes: *Request*, *Prognosis* and *Issue*. As can be seen from the CDL specification, the *Request_Part_Control* object is related to the *Part*, *Maintenance_Engineer* and *Warehouse* business objects. This business object can be in three states: '*initial*', '*processing*' or '*handled*'. The business process *Request* can change the state of the business object *Request_Part_Control* from '*initial*' to '*pro-*

cessing' on the basis of the incoming event *request*. Likewise, the *Prognosis* and *Issue* process can change the state of this control object, as described in section 2.1.

```
#include metamodel.cdl
entity Request_Part_Control {
  // cdl description of the control object "request_part_control"
  // this object comprises all three business processes and registers
  // the state of them (initial, processing or handled).
  // static aspects
  attribute workflow_ID, state_description, active_Process;
  relationship for Many Part inverse requested_in;
  relationship by One Maintenance_Engineer inverse requests;
  relationship from Many _Warehouse inverse has;
  // dynamic aspects
  state {initial, processing, handled}

  process Request {
    // process description here
    }; # end str
  }; # end process Request

  process Prognosis {
    attribute consumptionHistory, consumptionPeriod, expectedSpecialSale,
      totalExpectedConsQuantityinPeriod;
    relationship of Many Parts inverse has;
    relationship for Many Warehouse inverse has;
    event register_expected;
    event register_exptected_stock;
    void forecast(in partID, in stockID, in warehouseID, in consumptionPeriod,
                  in consumptionHistory);
    void manualReorderPointPlanning (in artID, in stockID, in warehouseID);
  }; # end process Prognosis

  process Issue_Part {
    attribute quantityRequest;
    event issue;
    apply StateTransitionRule IssueProcessing {
      trigger = {issue}
      source  = processing
      target  = handled
    }; # end str
  }; # end process Issue
}; # end entity Request_Part_Control
```

3. *Instantiating the Meta-CDL Enterprise Model:*
In this step, the CDL definitions given above are instantiated to a Meta-CDL-Model representing the enterprise. The Meta-CDL-Model is stored in the ConceptBase tool as already explained (see fig. 3), and can be reused each time the Meta-CDL Enterprise/Legacy Model need to be instantiated. The Meta-CDL model represents all CDL modeling constructs such as business objects and business processes and their constituents as already explained.
The next step is to to link the forward engineered model *Request-Part* to its CDL-Meta-Model in order to be able to map it later on to its reverse-engineered counterpart.
The Telos specification that follows is a textual representation of the graphical elements of the ConceptBase graph browser depicted in fig. 3.

```
Prognosis in Process with                          forecast in Operation with
        partof                                             owner
                x : Request_Part_Control                           o : Prognosis
        FOfeatures                                         usedAttrib
                f1 : consumptionHistory;                           a1 : partID;
                f2 : consumptionPeriod;                            a2 : stockID;
                f3 : ExpectedSpecialSale;                          a3 : warehouseID;
                f4 : totalExptectConsQuantityinPeriod;             a4 : consumptionPeriod;
                f5 : manualReorderPointPlanning;                   a5 : consumptionHistory
                f6 : forecast                              end
end
```

The two Telos frames above define features and operations of the process *Prognosis* as part of the BO *Request_Part*, and an operation *forecast* that is executed during this process (forecasting is used to determine the future consumption of a part). These are specified as instances of the Meta-CDL class *Process* and the Meta-CDL class *Operation*, respectively (see fig. 3). In fig. 3, it is shown that the *forecast* operation uses the attributes like *partID*, *warehouseID*,*stockID*, *consumptionPeriod ConsumptionHistory* to perform its objectives. In section 3.3 we will show how

the signature of this operation can be parameterized with components of a legacy operation signature.
After the Telos frames that are generated on the basis of the forward engineered CDL-descriptions and connected to the Meta-CDL-Model, we can proceed with the second phase in the *BALES* methodology: the Reverse Engineering Phase.

3.2 Reverse Engineering

1. *Reverse Engineered Model:*
   During this step the reverse engineered model is constructed as already explained in the previous. Reverse engineered legacy processes such as *Material_Recourse_Planning* (MRP) and *Purchase_Requisition* and wrapped objects like *Part*, *Plant*, *Warehouse*, etc., are represented in the reverse engineered model as shown in fig. 1. The legacy process *Material_Recourse_Planning* is used to determine the requirements for parts at a maintenance location.
2. *CDL-Specification of the Legacy Model:*
   We can now provide a CDL-specification on the basis of the reverse engineered model. As an example we use the interface of the legacy object *Warehouse*, see bottom part of fig. 2, whose interface is described below in CDL.
   As can be seen from this example the legacy object *Warehouse* encapsulates the legacy process *Material_Requirements_Planning*. This legacy process can be used to plan all the part requirements in the warehouse. For this purpose it uses the legacy operation *forecastDetModel*. The definitions in the LO *Warehouse* will subsequently be used as a basis to construct the interface of the business object *Warehouse*.

```
// Definition of the legacy entity: Warehouse

[keys={orderID}] entity Warehouse  {
  relationship ordered_for Many Part inverse ordered_by;
  relationship has Many Plant inverse of;

  attribute int plantID;
  [required] attribute String warehouse_name, warehouse_address, warehouse_place;
  state ordering{initial, planning, planned}

  // Definition of the Material Requirements Planning business process
  process Material_Requirements_Planning {
    // the relations of the process object, with other components
    relationship of References Part;
    relationship for References Plant;
    relationship in References Warehouse;

    // the dynamic behavior
    event register_expected;
    event start_long_term_planning;
    event start_stat_analysis;

    // Methods to implement MRP
    // forecast stock on basis of deterministic planning
    void forecastDetModel (in partID, in stockID, warehouseID,
                           in consumptionPeriod, in consumptionHistory);
    // forecast stock on basis of consumption based planning
    void planProduct (in artID, in stockID, in warehouseID);

    // state transition rule of MRP
    apply StateTransitionRule ProgProcessing {
      trigger = {register_exp_stock}
      source  = processing
      target  = handled
    }; # end str
  }; // end process Material_Requirements_Planning
};  // end Warehouse entity
```

3. *Linking the CDL specifications to the Meta-CDL Legacy Model:*
   After the CDL definitions are included in the ConceptBase repository, they are instantiated to the appropriate legacy components in the Meta-CDL Legacy-Model. For example, the legacy object *Warehouse* is instantiated from the Meta-CDL Legacy Modeling construct *L_Entity*. Another example is the legacy process *Material_Requirements_Planning* that is represented as instance of the legacy process

construct *L_Process*, see fig. 1. This object would look like as follows in Telos frame syntax:

```
L_Process Material_Requirements_Planning with
        partof
                x : Warehouse
        FOfeatures
                f1: consumptionPeriod;
                f2: consumptionHistory;
                f3: STR_ProgProcessing;
                f4: forecastModelDet
end
```

```
Operation forcastModelDet with
        owner
                o : Material_Requirements_Planning
        usedAttrib
                a1 : partID;
                a2 : warehouseID;
                a3 : stockID;
                a4 : consumptionPeriod;
                a5 : consumptionHistory
end
```

The above two Telos frames depict the legacy components *Material_Recourse_Planning* and *forcastDetModel* as instantiations of the Meta-CDL constructs *L_Process* and *Operation*, respectively. The *Material_Recourse_Planning* legacy process features two attributes (*ConsumptionPeriod* and *ConsumptionHistory*), a state-transition rule (called *STR_ProgProcessing*) and a method (*forecastDetModel*).

After both the forward and reverse engineered CDL descriptions have been specified by means of the Meta-CDL-Model in ConceptBase, the actual linking of business objects and processes to legacy objects and processes can take place.

3.3 Parameterizing: Specifying BOs via Cross-Interface Linkages

This phase indicates that business objects like *PartStock* (a business object that describes the statues of a part at the warehouse) and *StockLocation* (the location of the warehouse where the parts are physically stored, e.g., a shelve) are partly implemented by means of the legacy object *Stock*, see fig. 1. Hence, the interfaces of the business objects such as *PartStock* and *StockLocation* can be partially constructed by connecting them to the interfaces of the legacy object *Stock*. In reality there will also be a need to define auxiliary objects that are required to adjust the structure and behavior of the legacy objects to what is expected at the BO level. However, this procedure will not be discussed further in this paper due to reasons of brevity.

To parameterize BOs with legacy objects the following two tasks need to be performed in succession.

Query to find potentially mappable components: The first task in the linking phase consists of identifying potential legacy constructs that can be linked to related business constructs. We will illustrate the process of linking LO interfaces to BO interfaces by means of a ConceptBase query. The query ('OpWithSameAttributes' that is a specialization of the class *Operation*) results in a set of operations that share one or more attributes have an identical signature.

```
QueryClass OpWithSameAttributes isA Operation with
  retrieved_attribute
    owner : LOType
  parameter
    proto_op : Operation
  constraint
    con:
      $ ( forall a/AttributE (this usedAttrib a) ==>
        ( proto_op usedAttrib a))
      and
        ( forall b/AttributE (proto_op usedAttrib b) ==>
        ( this usedAttrib b)) $
end
```

Essentially the query determines those operations in the repository which are owned by a legacy object type and have the same signature (used attributes) as

the operation provided as parameter *proto_op* (signifying a prototypical object). The operation supplied as parameter to the query belongs to a business operation described in a Meta-CDL Business Model. This implies that we are looking for a legacy operation to match a business operation. A call *OpWithSameAttributes[forecast/proto_op]* of this query may yield the following result, which indicates a match between the BO operation *forecast* and the legacy operation *forecastDetModel*.

```
forecastDetModel in OpWithSameAttributes[forecast/proto_op] with
   owner
     fmdet_owner : Material_Requirements_Planning
end
```

As can be seen from the above result the legacy operation *forecastDetModel* is a candidate to implement the *forecast* business object operation. The answers obtained by queries are first checked against some simple type-safety criteria (mentioned below) and are also validated by an analyst to resolve semantic mismatches at the operation level. After successful validation, the interfaces of the *forecast* and *forecastDetModel* can be inter-linked.

Parameterize: The results of the queries to find potentially mappable components are used to create the interface specifications of the business objects. For this purpose we use the initial CDL specification for business objects (step-3) as described in fig. 2 where we connect business component specifications with references to equivalent (mappable) legacy component specifications that we identified by means of querying. An example of such a mapping is given below:

```
process Prognosis {
  attribute consumptionHistory, consumptionPeriod, expectedSpecialSale,
    totalExpectedConsQuantityinPeriod;
  event register_expected;
  event register_exptected_stock;
  // Mapping of forecasting method to legacy process component MRP
  this.forecast --> Warehouse.Material_Requirements_Planning.
          forecastDetModel(in partID, in stockID, in warehouseID,
                    in consumptionPeriod, in consumptionHistory);
  void manualReorderPointPlanning (in int artID, in stockID, in warehouseID);
}; # end process Prognosis
```

This example defines the business object operation *forecast* in terms of the legacy operation *Material_Requirements_Planning* which is embedded in the business object by means of the linking operator $\mapsto$.

In many cases it would be possible for methods at the business object level to be passed methods found at the legacy level as arguments or return legacy level functions as results. It is convenient to view such BO methods as *higher order functions* as they can accept legacy functions as parameters or return functions as results. This issue raises type safety problems as we may get runtime errors if we pass and subsequently invoke an inappropriate function from a high order function.

To ensure type safety on method arguments and method results we require the use of *argument contravariance* (expansion) and *result covariance* (restriction). Method results are said to covariant – they vary in the same way as function types. Result types must be more specific for the function type to be more specific. Conversely, argument types are said to contravariant - they vary in the opposite way as the function type. Argument types must be more general for the function type to be more specific [26]. We can informally explain this as follows. Assume we expect a function or method f to have type $t_1 \rightarrow t_2$, where t_1 are its arguments and t_2 its results. Therefore, we consider t_1 arguments as permissible when calling f. Now assume f actually has type $t'_1 \rightarrow t'_2$ with $t_1 \leq t'_1$, where $\leq$ is a special operator denoting a subclass to superclass relationship. Then we can pass all the expected permissible arguments of type t_1 without type

violation; f will return results of type t_2' which is permissible if $t_2' \leq t_2$ because the results will then also be of type t_2 and are therefore acceptable as they do not introduce any type violations. The subject of type safety regarding the parameterization of BOs by legacy counterparts is currently under research scrutiny.

4 Conclusions and Future Research

Enterprises need flexible, modular business processes that can easily be configured to meet the demands of business and technology changes. When developing applications based on business objects and processes it is important to address two factors: (a) requirements for change so that business information systems can evolve over time, and (b) the linking of business objects with legacy information systems. The common aim of these two requirements is the ability to combine new and existing (legacy) business components within a running application. In both cases there is a need for the added business components to interoperate seamlessly with components present in the current execution environment. This should happen without the risk of disrupting the application or business process it models, thus, facilitating the graceful, incremental, evolution of complex systems.

In this paper we have described the *BALES* (binding Business Application objects to LEgacy Systems) methodology that we are currently developing. This methodology has as its main objective to inter-link parameterizable business objects to legacy objects. Legacy objects serve as conceptual repositories of extracted (wrapped) legacy data and functionality. These objects are, just like business objects, described by means of their interfaces rather than their implementation. Business objects in the *BALES* methodology are configured so that part of their implementation is supplied by legacy objects. This means that their interfaces are parameterizable (or self-describing) to allow these objects to evolve by accommodating upgrades or adjustments in their structure and behavior.

The results that we have presented are core results in nature. Extensions are needed in several directions to guarantee a practical methodology. For example, problems with regard to granularity need to be solved in a more efficient manner. As can be observed by the 'Request-Part' enterprise workflow, in section 3, the granularity of the legacy system was higher than that of the business model counterpart. Currently, we use simple decomposition techniques to solve this problem. In addition to this, problems in connection with type safety need to be further investigated. In its current form the methodology does not provide yet mechanisms to bind organizational policies to business processes and objects. Research work reported in [16] and [28] seems to be particularly useful for this purpose. Lastly, the evolution of business processes can be compared to software components configuration management [29]. We plan to combine some of the ideas that have been developed in this area with the approach presented herein to in order to accommodate pro-active behavior in our mapping methodology.

References

[1] D.A. Taylor. *Business Engineering with Object Technology*. John Wiley and Sons, Inc., New York, 1995.

[2] M. Hammer and J. Champy. *Re-engineering the Corporation: A Manifesto for Business Revolution*. Harper Collins, New York, 1993.

[3] M.L. Brodie. "The Emperor's Clothes are Object-Oriented and Distributed" in M.P. Papazoglou and G. Schlageter, editors, *Cooperative Information Systems: Trends and Directions*, Academic Press, 1998.
[4] S. Abinavam and et al. *San Francisco Concepts & Facilities*. International Technical Support Organization, IBM, February 1998. SG24-2157-00.
[5] M. L. Brodie and M. Stonebraker. *Migrating Legacy Systems: Gateways, Interfaces and the Incremental Approach*. Morgan Kaufman Publishing Company, 1995.
[6] A. Umar. *Application (Re)Engineering: Building Web-based Applications and Dealing with Legacies*. Prentice Hall, New Jersey, 1997.
[7] R.C. Aronica and D.E. Rimel Jr. "Wrapping your Legacy System", *Datamation*, 42(12):83–88, June 1996.
[8] P. Robertson. "Integrating Legacy Systems with Modern Corporate Applications", *Communications of the ACM*, 50(5):39–46, 1997.
[9] I. Jacobson, M. Griss, and P. Jonsson. *Software Reuse: Architecture, Process and Organization for Business Success*. Addison Wesley, 1997.
[10] T. Curran, G. Keller, and A. Ladd. *SAP R/3 Business Blueprint: Understanding the Business Process Reference Model*. Prentice-Hall, New-Jersey, 1998.
[11] A. W. Scheer. *Business Process Engineering: Reference Models for Industrial Enterprises*. Springer-Verlag, 1994.
[12] M.P. Papazoglou and W.J. van den Heuvel. "From Business Processes to Cooperative Information Systems: An Information Agents Perspective", in M. Klusch, editor, *Intelligent Information Agents*, Springer-Verlag, Feb. 1999.
[13] P. Eeles and O. Sims. *Building Business Objects*. John Wiley & Sons, New York, 1998.
[14] I. Graham. *Migrating to Object Technology*. Addison-Wesley Publishing Company, Workingham, England, 1994.
[15] I. Jacobson and M. Ericsson. *The Object Advantage: Business Process Re-engineering with Object Technology*. ACM Press, Addison-Wesley Publishing Company, Workingham, England, 1995.
[16] E. Yu. *Modeling Strategic Relationships for Process Engineering*. PhD thesis, University of Toronto, 1994.
[17] Department of Defense Netherlands. "Methodiek voor het inrichten van de informatievoorziening op basis van bouwstenen ten behoeve van het ministerie van defensie", Technical Report, Defense Telematics Organization, 1997.
[18] M.P. Papazoglou and S. Milliner. Content-based Organization of the Information Space in Multi-database Networks", in B. Pernici and C. Thanos, editors, *Procs. CAISE'98 Conf., Pisa, Italy*, Springer-Verlag, 1998.
[19] Data Access Technologies. Business object architecture (BOA) proposal. BOM/97-11-09, OMG Business Object Domain Task Force, 1997.
[20] W. Hordijk, S. Molterer, B. Peach, and Ch. Salzmann. "Working with Business Objects: A Case Study", OOPSLA'98 Business Object Workshop, http://jeffsutherland.org/oopsla98/molterer, October 1998.
[21] J. Mylopoulos, A. Borgida, M. Jarke, and M. Koubarakis. "Telos: Representing Knowledge about Information Systems", *ACM Transactions on Information Systems*, 8(4):325–362, 1990.
[22] M.A. Jeusfeld, M. Jarke, H.W. Nissen, and M. Staudt. "ConceptBase: Managing Conceptual Models about Information Systems", in P. Bermus, K. Mertins, and G. Schmidt, editors, *Handbook on Architectures of Information Systems*. Springer-Verlag, 1998.
[23] M.P. Papazoglou and N. Russell. "A Semantic Meta-modeling Approach to Schema Transformation", In *CIKM-95: Int'l. Conf. on Information and Knowledge Management, Baltimore, Maryland*, 1995.
[24] M.P. Papazoglou and W.J. van den Heuvel. "Leveraging Legacy Assets", to appear in M. Papazoglou, S. Spaccapietra, Z. Tari, editors, *Advances in Object-Oriented Modeling*, MIT-Press, 1999.
[25] S. Chen. *Retrieval of Reusable Components in a Deductive, Object-Oriented Environment*. PhD thesis, RWTH Aachen, Information Systems Institute, 1993.
[26] B. Meyer. *Object-Oriented Software Construction*. Prentice-Hall, Englewood Cliffs, 1994.

[27] J. Mylopoulos. “Conceptual modeling and Telos”, in P. Loucopoulos and R. Zicari, editors, *Conceptual Modeling, Databases and Case: an Integrated View on Information Systems Development*, J. Wiley, New York, 1992.

[28] P. Kardasis and P. Loucopoulos. “Aligning Legacy Information Systems to Business Processes”, in B. Pernici and C. Thanos, editors, *Procs. CAISE’98 Conf., Pisa, Italy*, Springer-Verlag, 1998.

[29] M. Jarke, M.A. Jeusfeld, A. Miethsam, and M. Gocek. “Towards a Logic-based Reconstruction of Software Configuration Management”, *7th Knowledge-Based Software Engineering Conference*. IEEE Computer Society Press, Los Alamitos, California, 1992.

[30] Leonid Kalinichenko. “Workflow Reuse and Semantic Interoperation Issues”, in: *Workflow Management Systems and Interoperability* (A. Doğaç, L. Kalichenko, M.T. Özsu and A. Sheth eds.), NATO ASI Series, Springer, 1998