TOP-DOWN ENTERPRISE APPLICATION INTEGRATION WITH REFERENCE MODELS

Willem-Jan van den Heuvel & Wilhelm Hasselbring & Mike Papazoglou
Infolab, Dept. Information Management and Computer Science, Tilburg University,
PO Box 90153, NL-5000 LE Tilburg, Netherlands,
Email: { wjheuvel|hasselbring|mikep } @kub.nl

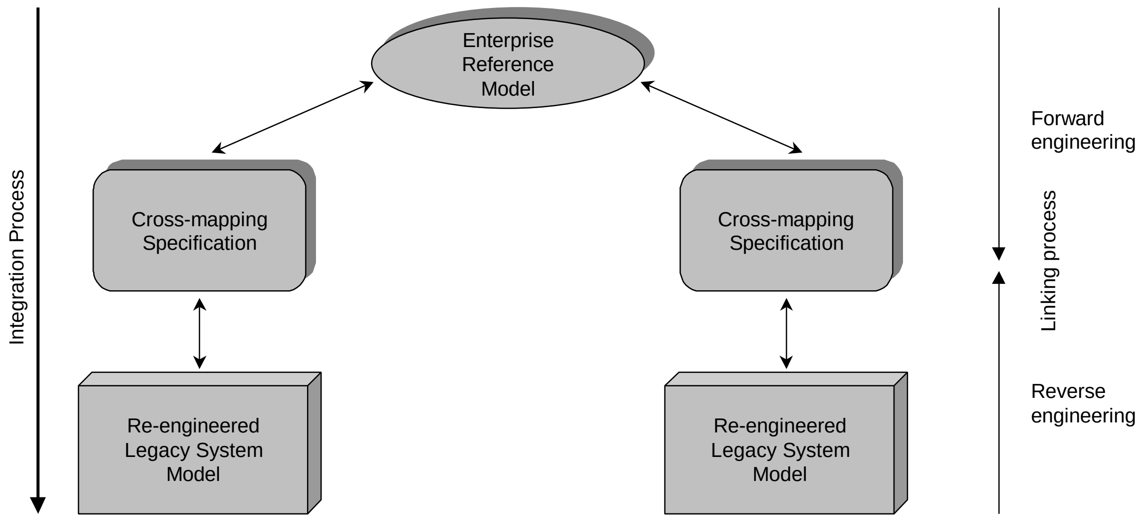
ABSTRACT

For Enterprise Resource Planning (ERP) systems such as SAP R/3 or IBM SanFrancisco, the tailoring of reference models for customizing the ERP systems to specific organizational contexts is an established approach. In this paper, we present a methodology that uses such reference models as a starting point for a top-down integration of enterprise applications. The re-engineered models of legacy systems are individually linked via cross-mapping specifications to the forward-engineered reference model's specification. The actual linking of reference and legacy models is done with a methodology for connecting (new) business objects with (old) legacy systems.

INTRODUCTION

With the traditional bottom-up approach to the integration of existing (legacy) systems, the structure of the (merged) integrated information models is highly determined by the overlaps among the component system models. As discussed in Hasselbring (1999), the maintenance of such integrated models may become a serious problem, because the merged models rapidly become very complex; usually more complex than required for the actual integration goals. This situation can lead to severe scalability problems with respect to execution performance, usability, and maintenance. For a discussion of the resulting problems refer to Hasselbring (1999).

Another way to approach the integration of heterogeneous information systems is a *top-down* process. Starting with common reference models, the individual component models are integrated into these common reference models Hasselbring (1999). The resulting integration process is illustrated in Figure1. The local models of the legacy systems are not integrated into a common global model (which would be the 'federated schema' in federated database systems (Sheth & Larson 1990)) as it would be the case with the traditional bottom-up approach. Instead, an integration of the given reference model with each individual local model is constructed via a linking mechanism (a form of type matching in our methodology). A cross-mapping specification defines the mapping from the (given) reference model to the (local) legacy models. The *integration* process starts top-down with the reference model. The *linking* process combines forward and reverse engineering techniques. Both, the reference model and the legacy models are specified in our Component Definition Language (CDL), before they are integrated. An important difference with integration in federated *database* systems is that with enterprise applications, we integrate business models, not database schemas. CDL has been proposed as the standard component specification language by the Business Object Domain Task Force of the OMG. It is a superset of the OMG Interface Definition Language (IDL) and the ODMG Object Definition Language (ODL). We introduce specific extensions for business modeling and, in particular, cross-mapping specification.



Enterprise Reference Model
Forward engineering
Cross-mapping Specification
Cross-mapping Specification
Integration Process
Linking process
Reverse engineering
Re-engineered Legacy System Model
Re-engineered Legacy System Model

Figure 1: Top-down integration of legacy systems with a reference model for enterprise applications as starting point for the integration.

For the actual linking of reference and legacy models, we employ a methodology for business model integration, called Business Applications to LEgacy Systems (*BALES*) (van den Heuvel 2001). This methodology allows reusing as much of the legacy data and functionality as needed for the development of applications that meet modern organization requirements and policies. In particular, the *BALES* methodology allows to construct configurable business applications on the basis of business objects and activities that can be linked through parameterization to their legacy counterparts.

THE BALES METHODOLOGY FOR LINKING BUSINESS MODELS

Most of the approaches to integrate legacy systems with modern applications are designed around the idea that data residing in a variety of legacy database systems and applications represents a collection of entities that describe various elements 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 activities and policies which need to be reflected in the legacy systems. Although part of the functionality of a legacy system can be readily used, many of its business activities and policies may change with the passage of time.

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 (Eeles & Sims 1997). Business object 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.

The *BALES* methodology, that is under development (van den Heuvel 2001), has as its main objective to link business objects (BOs) 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 within the reference model can thus have a part that is directly supplied from some legacy data and services which it integrates with data and services defined at its own level. This means that the business object interfaces are parameterizable to allow these objects to evolve by accommodating upgrades or adjustments in their structure and behavior.

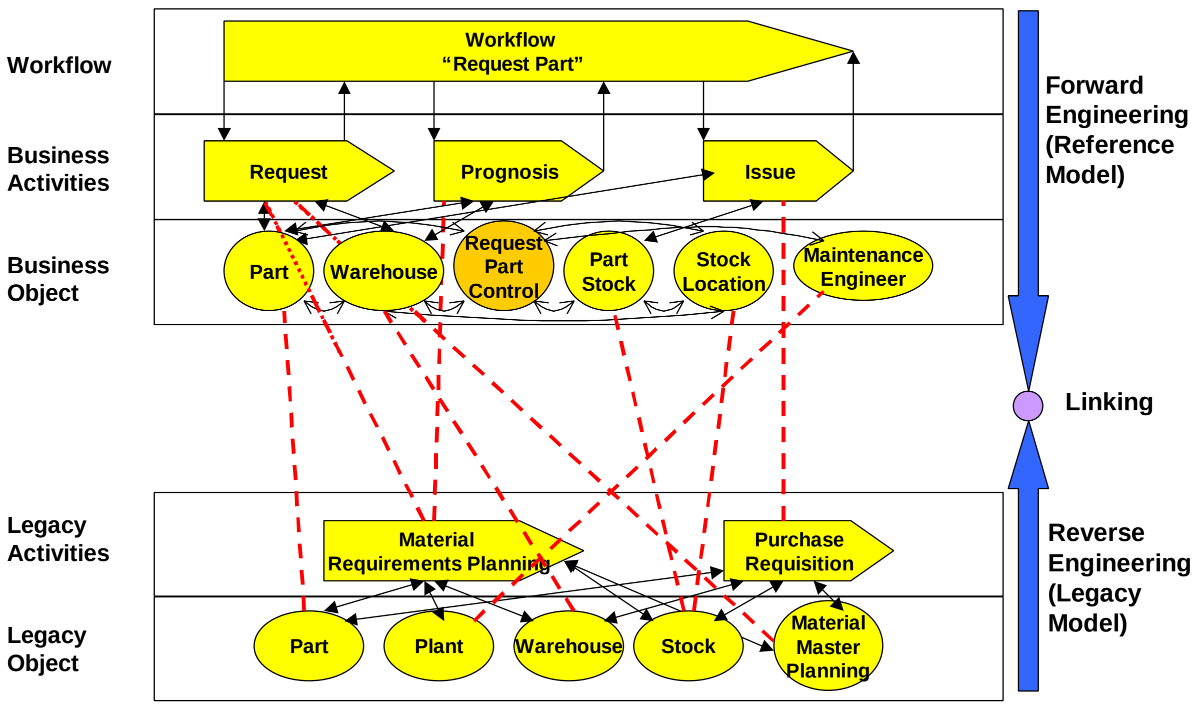


Figure 2: Methodology of linking reference and legacy models. Workflows initiate business activities and business activities use business objects. The dashed lines are meant to illustrate (possible) mappings between reference and legacy elements.

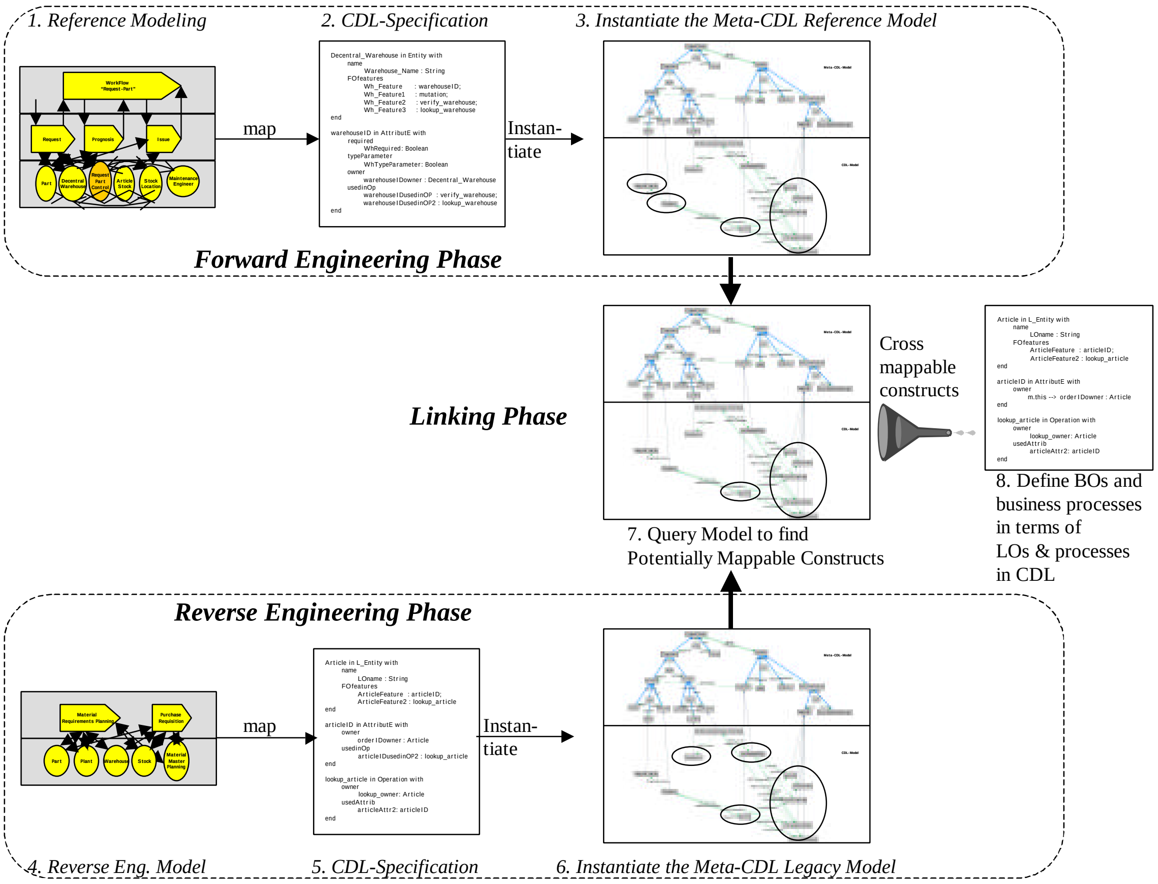
The core of the *BALES* linking methodology comprises three phases, as illustrated on the right border of Figure2: *forward engineering*, *reverse engineering* and *linking*. To illustrate this linking methodology, a simplified example is drawn from the domain of maintenance and overhaul of aircrafts. This example is inspired from building block definitions that are currently developed at the Department of Defense in the Netherlands (DoDN 1997).

The upper part of Figure illustrates the reference model for the business domain (which is based on the Defense and Aerospace reference models for SAP R/3 (SAP 2000)) in terms of workflows, business activities and business objects. As can be seen from this figure, the reference model defines a `Request_Part` workflow which comprises three business activities: `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 a request:

1. 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` activity 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` activity uses the business (entity) objects *Part* and `Warehouse` for this purpose. Subsequently, the workflow initiates the `Issue` activity (see Figure). The `Issue` activity registers administrative results regarding the dispatching of requested parts 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 activities. 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.

2. 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 activity first registers the request information provided by the *Request* business activity and runs a prognosis on the basis of the availability and consumption history of the requested part. The `Prognosis` activity uses information from the `Part` and `Warehouse` business objects for this purpose. After the prognosis finished successfully, the part can be reserved. If the results of the activity `Prognosis` are negative with respect to the future availability of the requested aircraft part, another workflow for ordering parts is activated.

The lower part of Figure represents the result of the reverse engineering activity in the form of two activities (wrapped applications and related databases) `Material_Requirements_Planning` and `Purchase_Requisition`. These activities make use of five legacy objects to perform their operations. Figure also indicates that the reference workflow draws not only on "modern" business objects and activities, but also on existing (legacy) data and functionality to accomplish its objectives. For example, business activities such as `Request` and `Issue` on the reference model level are linked to the legacy activities `Material_Requirements_Planning` and `Purchase _Requisition` by means of solid lines. This signifies the fact that the activities on the reference model level *reuse* the functionality of the activities at the legacy model level. The same applies for business objects at the reference 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 reference and legacy models have already been resolved.



```
Decentral_Warehouse in Entity with
        name
                Warehouse_Name : String
        FOfeatures
                Wh_Feature      : warehouseID;
                Wh_Feature1     : mutation;
                Wh_Feature2     : verify_warehouse;
                Wh_Feature3     : lookup_warehouse
end

warehouseID in AttributE with
        required
                WhRequired: Boolean
        typeParameter
                WhTypeParameter: Boolean
        owner
                warehouseIDowner : Decentral_Warehouse
        usedinOp
                warehouseIDusedinOP  : verify_warehouse;
                warehouseIDusedinOP2 : lookup_warehouse
end
```

```
Article in L_Entity with
        name
                LOname : String
        FOfeatures
                ArticleFeature  : articleID;
                ArticleFeature2 : lookup_article
end

articleID in AttributE with
        owner
                m.this --> orderIDowner : Article
end

lookup_article in Operation with
        owner
                lookup_owner: Article
        usedAttrib
                articleAttr2: articleID
end
```

```
Article in L_Entity with
        name
                LOname : String
        FOfeatures
                ArticleFeature  : articleID;
                ArticleFeature2 : lookup_article
end

articleID in AttributE with
        owner
                orderIDowner : Article
        usedinOp
                articleIDusedinOP2 : lookup_article
end

lookup_article in Operation with
        owner
                lookup_owner: Article
        usedAttrib
                articleAttr2: articleID
end
```

Figure 3: The *BALES* linking methodology.

Figure illustrates the integration approach of the *BALES* methodology and the individual steps applied during its three phases. The forward engineering phase transforms a conceptual reference model (e.g., SanFrancisco (Abinavam et al. 1998) Reference Models specified in the UML notation (Booch et al. 1999)) 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 activities which correspond to steps 1, 2, and 3 in Figure. In the second phase of the *BALES* methodology, we represent the legacy objects and activities in terms of CDL and link them to a Meta-CDL Legacy Model. The activities during the reverse engineering phase, which correspond to steps 4, 5, and 6 in Figure 3, are similar to those performed during the forward engineering phase. The actual linking is then done in step 7, and the cross-mapping specification is constructed in the final step 8. Below, we illustrate the BALES methodology for constructing a cross-mapping specification by means of the aircraft maintenance and overhaul example following those steps:

1. Reference Modeling:

The forward engineering activity starts with the given reference model. The reference model reveals the activities, structure, information, actors, goals, and constraints of the business in terms of business objects, activities, and workflows, and is illustrated by the reference workflow in the upper part of Figure 2.

2. CDL-Specification of the Reference Model:

The interface descriptions of the business objects and activities need to be constructed on the basis of the reference model. To formally describe the interfaces of business objects, we use a variant of the CDL that has been developed by the OMG (DAT 1997). CDL is a declarative specification language — a superset of the OMG Interface Definition Language (IDL) and the ODMG Object Definition Language (ODL) with specific extensions for business modeling and, in particular, cross-mapping specification. A specification in CDL defines business object interfaces, structural relationships between business objects, collective behavior of related business objects, and temporal dependencies among them (DAT 1997). Detailed descriptions of the CDL syntax can be found in DAT (1997) and some practical experience with the use of CDL is discussed in Hordijk (1998).

The reference model represented in the upper part of Figure 2, serves as a starting point to specify the business object/activity in CDL. Figure 4 gives an extract of the CDL specification involving a business object with interesting dynamic behavior, namely the `Request_Part_Control` object. This CDL specification describes the interface of the business control object `Request_Part _Control` (see Figure 2) and shows that this business object encapsulates three business activities: `Request`, `Prognosis`, and `Issue`. The dynamic behavior of the encapsulated `Prognosis` activity should be interpreted as follows: the `Prognosis` activity is triggered by the incoming signal `register_expected_stock`. After this signal is received, the activity moves from the state `initial` to the state `forecasting`. This is expressed with the state transition rule (STR) `Start_forecasting`. While the `Prognosis` activity resides in the state `forecasting`, it can perform the `forecast` operation. This operation stochastically estimates the required stock on the basis of past data (stock levels) in the warehouse (`warehouseID`) and required future demand of the part (`partID`) for a specific period (`consumptionPeriod)`.

The `manualReorderPlanning` business operations of the `Prognosis` activity refers to the situation where the user has to define the reorder point and the safety stock him/herself. This approach is in contract to the automatic reorder planning (method where both parameters are automatically forecasted).

Reorder planning is a special category consumption-based planning that calculates the reorder point on the basis of past and future consumptions, delivery lead times, etc. In this planning strategy, the available stock is compared with the reorder point; if the actual stock gets below the reorder point, the system automatically creates an order form (Keller, G. & Teufel, T. 1998).

3. Instantiating the Meta-CDL Reference Model:

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. For this purpose we utilize the ConceptBase system (Jeusfeld et al. 1998), which 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 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.

```
entity Request_Part_Control
// this object comprises all three business activities and registers
// the state of them (initial, processing or handled).
...
  activity Prognosis
    relationship boRPC  IsPartOf  Request_Part_Control inverse bpP ;
    relationship wflR  Many 0..* Request_art_Workflow inverse bpP ;
    attribute String consumptionHistory;
    attribute Date consumptionPeriod;
    attribute Integer expectedSpecialSale;
    attribute Integer totalExpectedConsQualityinPeriod;
    Void manualReorderPlanning (in Integer partID, in Integer stockID,
      in Integer warehouseID);
    signal register_expected_stock ();
    states prognosis (initial, manual_planning, forecasting, stopped);
    during (prognosis==forecasting) withForecast
    attribute Integer partID;
    attribute Integer stockID;
    Void forecast (in Integer partID, in Integer stockID, in
      Integer warehouseID, in Date consumptionPeriod,
      in String  consumptionHistory);
    ;
    apply StateTransitionRule Start_forecasting
    trigger = register_expected_stock();
      source = initial;
      target = forecasting
      ;
  ; // End: Prognosis
  activity Issue_Part  .. ;
```

Figure 4: The CDL specification for the reference business (entity) object `Request_Part_Control`.

After the interfaces of the business objects and activities have been specified in CDL, the CDL specifications are instantiated according to a Meta-CDL reference model. This meta model depicts the *instantiations* of the CDL model elements. It defines how the CDL constructs are related to each other, and provides information about their types. The CDL meta-modeling step is used as a basis to infer how the constructs found in a Meta-CDL reference model can be connected to related constructs found in the legacy models (see below). In summary, the Meta-CDL model serves as a shared description (could be compared to the 'canonical data model' in federated database systems (Sheth & Larson 1990)) to which the forward and the reverse engineered CDL models will be linked in order to ascertain which (portions of) legacy elements can be linked to the reference model level. In this way, it is possible to parameterize reference model elements with related legacy elements for linking them to each other in the cross-mapping specification.

4. Reverse Engineered Legacy 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 (Papazoglou & Russel 1995, Papazoplou & van den Heuvel 2000). The legacy object model comprises a distinct legacy object and activity layer in the BALES methodology (see bottom part of Figure 2).

Reverse engineered legacy activities such as `Material_Requirements_Planning` and `Purchase_Requisition` and wrapped objects like `Part`, `Plant`, `Warehouse`, etc., are represented in the reverse engineered model as illustrated in Figure 2. The legacy activity `Material_Requirements_Planning` is used to determine the requirements for parts at an aircraft maintenance location.

```
Legacy_entity Warehouse
  relationship ordered_for Many Part inverse ordered_by;
  relationship has Many Plant inverse of;
  attribute Integer plantID;
  [required] attribute String warehouse_name, warehouse_address,
     warehouse_place;
  state orderinginitial, planning, planned

  // Definition of the Material Requirements Planning legacy process
  Legacy_activity Material_Requirements_Planning
    relationship loP  Many 0..* Part inverse mrpLpP ;
    relationship loW  Many 0..* Warehouse inverse mrpLpW ;
    relationship loPl  Many 0..* Plant inverse mrpLpPl ;
    attribute String consumptionHistory;
    attribute Date consumptionPeriod;
    attribute Integer totalExpectedConsQualityinPeriod;

    signal register_expected ();
    signal register_expected_stock ();
    Void forecastStochModel (in Integer partID, in Integer stockID, in
    Integer warehouseID, in Date consumptionPeriod,
    in String  consumptionHistory);
    states prognosis (initial, manual_planning, forecasting, stopped);

    apply StateTransitionRule START_FORECASTING
     trigger = register_expected_stock();
     source = initial;
```

Figure 5: The CDL specification for the legacy (entity) object `Warehouse`.

5. CDL Specification of the Legacy Model:

The interfaces of the legacy objects and activities are described in CDL in the same way as we explained for reference activities and objects. Figure 5 presents an example interface of the legacy object `Warehouse`. As can be seen from this example, the legacy object `Warehouse` offers the legacy activity `Material_Requirements_Planning`. This legacy activity can be used to plan all the part requirements in the warehouse. For this purpose it uses the legacy operation `forecast-StochModel` (where stochastic planning is a special form of consumption-based planning, see the business operation `forecast`).

The definitions in the legacy object `Warehouse` will subsequently be used as a basis to construct the interface of the reference business object `Warehouse`.

6. Instantiating the Meta-CDL Legacy Model:

After the CDL specifications of the legacy components are available, they are also instantiated into the meta-model repository.

7. Link Phase of the CDL Meta Models:

When both the forward and reverse engineered CDL descriptions have been instantiated by means of the Meta-CDL model in ConceptBase, the actual linking of business objects and activities to legacy objects and activities can take place. At first, a measure of similarity between a given CDL description of the reference model I_0 and (a set of) CDL description(s) of the legacy system(s) $I_1 \cdots I_n$ is calculated. This measure is used to identify (part of) the CDL description of the legacy components in the set that fit(s) best to the given CDL definition of the reference (enterprise) model. In step 8 (see below), we actually define the (partial) mapping based on these measurements.

The algorithm to calculate the similarity of two (or more) CDL specification deals with type conformance issues like covariance (e.g., of the method results) and contra-variance (e.g., of method arguments). However, it leaves semantic integration of both schemes outside the scope.

Semantic interoperability has been extensively investigated in the field of (federated) database systems, f.e., in the context of data scheme integration. Most solutions comprise (a combination of) ontologies, meta-data or contexts (Kashyap 1998). Semantic integration of a reference model and a reverse engineered model of a legacy system is even harder as the terminology used in both specifications is inherently totally different: reverse engineered CDL code often includes obscure, "empty" names derived from code variables, whereas the reference model is mostly stated in (standardized) business terminology. For example, the legacy variable DSAG_TAS_LDD that has automatically been reverse engineered into a legacy attribute of a legacy object DSAG_TAS in a legacy model is semantically equivalent to the attribute `product_name` of the business object `Product` of a corresponding reference model.

Therefor, we propose to firstly compare type skeletons, that we call *type trees* (see Figure 6), of both legacy and reference objects. Later on, we will resolve any remaining semantic mismatches. In a type tree, a leaf represents a type, and a node represents a typed entity. A type-tree is a directed acyclic graph $G=(N,E,S,C,R)$, where N is the set of nodes, and E is the set of edges. The leaves $L \subseteq N$ of G are nodes, with no outgoing edges. The leaves are marked by *Simple Type*, all other nodes by *Composite Type*. The inner nodes in $E - L$ are called *constructor nodes*. The predefined set *Simple Types* includes types as `Integer`, `String`, etc. The predefined set *Composite Types* includes type constructors like `operation` and `during`. Both sets may be enhanced by user defined types. One can assign a role to some edges, e.g., for a node with the constructor `method`, `attribute`, `return type`, `input type`, and `output type`. Both sets may be extended by user defined types. The `function S: L ➔Simple-Types` maps a simple type to each leaf. The function `C: E-L ➔ Composite Types` assigns to each constructor node a composite type. The partial function `R: E ➔ Roles` can be used to assign a role to an edge (Heuvel, W.-J. van den & Reussner, R. 2000).

Given two type trees t_1, t_2 the algorithm computes a number, which is a measurement for the similarity of t_1 and t_2 . The higher this number, the greater the similarity. The algorithm is implemented as a set of active rules and queries to compare the (linked) legacy and reference models that are stored in the ConceptBase system.

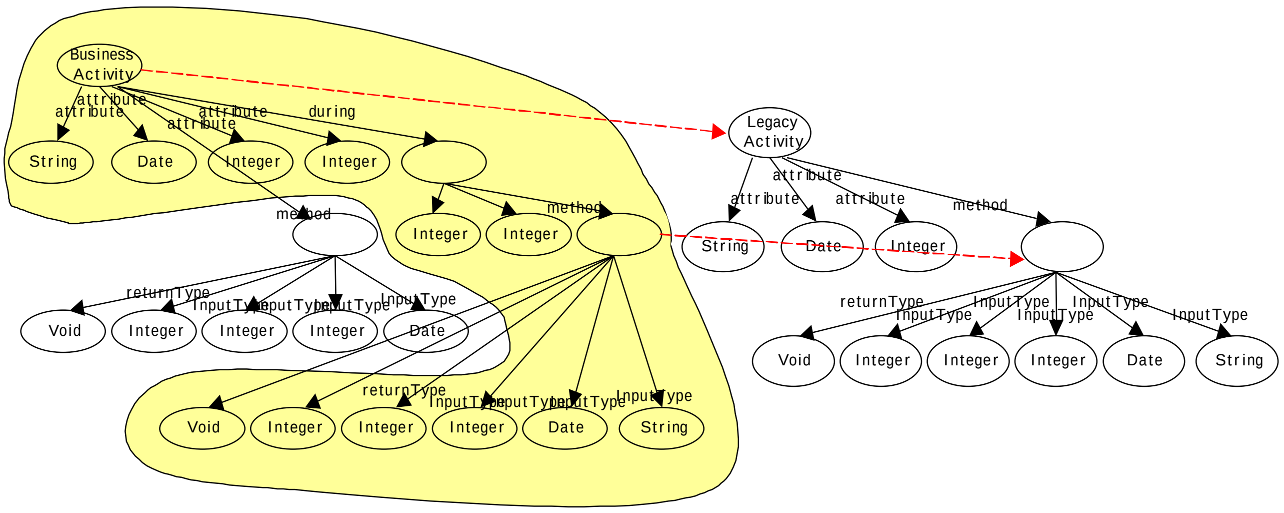


Figure 6: The type-tree matching process for the reference model and the legacy enterprise model.

Figure 6 illustrates the result of the type-tree matching process for the CDL examples. At the left hand side of this figure, the type tree of the reference (enterprise) model is depicted. The right hand side expresses the reverse engineered (SAP R/2) legacy system model, and has been derived from Figure 5. The algorithm now calculates the match between (parts of) both type trees. The result is graphically expressed as the grey area at the left hand side of Figure 6; the arrows point to the constructs that we compared.

The linking algorithm results in an equivalence rate of 139/180 (≈77%) for the business activity `Prognosis` and the legacy activity `Material_Requirements_Planning`. The equivalence rate of the business operation `forecast` with the legacy operation `forecastStochModel` is obviously 1 (thus a full match). For details on the algorithm, we refer the reader to van den Heuvel (2001).

8. CDL Specification of the Cross Mapping:

The *BALES* methodology results in a CDL specification of reference elements in terms of their related legacy counterparts. The interfaces that are most likely to match according to the linking algorithm, now need to be checked by the designer to resolve semantic conflicts. The syntactically and semantically matched constructs thereafter need to be specified in the resulting parameterized business (/task) object(s). For this purpose we use the initial CDL specification for reference objects from step 3, in which we connect reference element specifications with links to equivalent (mappable) legacy component specifications that we identified by means of the matching algorithm.

An example of such a mapping is given in Figure 7. This (simplified) example defines the reference object operation `forecast` in terms of the legacy operation `forecastStochModel` (part of the

Material_Requirements_Planning legacy activity/program) which is embedded in the business object by means of the linking operator -->.

This linking process is followed for each legacy system that shall be integrated into the common reference model.

```
activity Prognosis
    relationship boRPC  IsPartOf  Request_Part_Control inverse bpP ;
    relationship wflR  Many 0..* Request_art_Workflow inverse bpP ;
    attribute String consumptionHistory;
    attribute Date consumptionPeriod;
    attribute Integer expectedSpecialSale;
    attribute Integer totalExpectedConsQualityinPeriod;
    Void manualReorderPlanning (in Integer partID, in Integer stockID,
    in Integer warehouseID);
    …
    during (prognosis==forecasting) withForecast
      attribute Integer partID;
      attribute Integer stockID;
      // Mapping of forecasting method to legacy process component MRP
      Void this.forecast --> Warehouse.Material_Requirements_Planning.
      forecastStochModel(in Integer partID, in Integer stockID, in Integer
      warehouseID, in Date consumptionPeriod, in String consumptionHistory);
    ;

    apply StateTransitionRule Start_forecasting
     trigger = register_expected_stock();
     source = initial;
     target = forecasting
```

Figure 7: The CDL specification for the reference business object's operation forecast in terms of the legacy operation forecastStochModel.

CONCLUSIONS AND FUTURE RESEARCH

In this paper, we present a top-down approach to enterprise application integration, whereby reference models are used as starting point for the integration process. The linking of reference and legacy models combines forward and reverse engineering techniques employing the BALES methodology. A resulting cross-mapping specification defines the mapping from the reference model to the individual legacy models. The BALES 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. Future research includes considering similarity weights in the matching algorithm.

The reference models serve as the starting point in the integration process with the top-down approach. The overall integrated system will be more scalable than with the bottom-up approach, because the integrated reference models do not grow linearly with the number of component systems. The decentralized responsibility for maintaining the cross-mapping specifications reduces the central coordination needs and distributes the maintenance cost. Starting with the reference models should avoid changes to the fundamental structure of these models, making the integration more usable and scalable. For integrations with a small number of local/component systems, the top-down approach may not offer the optimal solutions, but for integrations with a large number of connected systems, we will obtain a more usable and maintainable overall systems architecture.

In practice, we can also expect a *yo-yo* approach, as discussed in Hasselbring (1999): the integration process alternates with bottom-up and top-down steps. For instance, the bottom-up process may provide input for extending the reference models. In the presented example, that was taken from a project with the Department of Defense in the Netherlands, it turned out that the existing reference models (the so called Defense and Aerospace Solution Maps (SAP 2000)) for SAP R/3 did not cover all requirements of this Dutch setting. These problems are currently addressed by SAP through extending their respective reference models according to these additional requirements. The development process must take feedback, which is based on experience with actual applications, into consideration. Anyway, it is important to *start* at the top (with the common models). To take the analogy of the yo-yo toy: when the game starts, the reel should be coiled up.

REFERENCES

Abinavam, S. et al. (1998) San Francisco Concepts & Facilities. Technical Report SG24-2157-00, IBM International Technical Support Organization, 1998.

Booch, G. et al. (1999) *Unified Modeling Language User Guide*. Object Technology Series. Addison-Wesley, Reading, MA, 1999.

DAT (1997) Data Access Technologies, OMG Business Object Domain Task Force,. *Business object architecture (BOA*) proposal BOM/97-11-09, 1997.

DoDN (1997) Department of Defense Netherlands. *Methodiek voor het inrichten van de informatievoorziening op basis van bouwstenen ten behoeve van het ministerie van defensie*, 1997.

Eeles, P. & Sims, O. (1997) P *Building Business Objects*. John Wiley & Sons, New York, 1998.

Hasselbring, W. (1999) Top-down vs. bottom-up engineering of federated information systems. In *Engineering Federated Information Systems (Proc. EFIS'99)*, pages 131–138. Infix-Verlag, 1999.

Heuvel, W.-J. van den & Reussner, R. (2000) Matching Component Interfaces, Technical Report Infolab, Tilburg University, The Netherlands and Interner Bericht at the Department of Informatics, University of Karlsruhe, Germany

Hordijk, W. et al. (1998) Working with business objects: A case study. In *OOPSLA'98 Business Object Workshop IV*. Springer-Verlag, 1998.

Jeusfeld, M. et al. (1998) ConceptBase: managing conceptual models about information systems. In P. Bermus, K. Mertins, and G. Schmidt, editors, *Handbook on Architectures of Information Systems*, pages 265–285. Springer-Verlag, 1998.

V. Kashyap and A. Sheth (1998) Semantic Heterogenity in Global Information Systems: the Role of Metadata, Context and Ontologies. In Papazoglou and Schlageter, editors, Cooperatve Information Systems, Academic Press, 1998.

Keller, G. & Teufel, T. (1998) SAP R/3 Process-Oriented Implementation, Addison Wesley Longman, 1998

Papazoglou, M.P & Russell, N.(1995) A semantic meta-modeling approach to schema transformation. In N. Pissinou, A. Silberschatz, E. K. Park, and K. Makki, editors, *CIKM-95: Int'l. Conf. on Information and Knowledge Management*, Baltimore, Maryland, 1995. ACM Press.

Papazoglou, M.P. & van den Heuvel, W.-J. (2000) Leveraging legacy assets. In M. Papazoglou, S. Spaccapietra, and Z. Tari, editors, *Advances in Object-Oriented Modeling*. MIT-Press, 2000. (in press).

SAP (2000) *SAP Aerospace & Defense*. available at `http://www.sap.com/products/industry/aero/`.

Sheth, A. & Larson, J. 1990) Federated database systems for managing distributed, heterogeneous, and autonomous databases. *ACM Computing Surveys*, 22(3):183–236, 1990.

van den Heuvel, W.-J. (2001) *Integrating Business Applications with Legacy Systems*. PhD thesis, Tilburg University, 2001. (in preparation).