A Methodology to support Web-services Development using Legacy Systems

Willem-Jan van den Heuvel
Infolab, Dept. of Info Systems and Info Management
Tilburg University
PO Box 90153
Tilburg
The Netherlands
wjheuvel@kub.nl

Jos van Hillegersberg
Dept. of Info Sciences
Erasmus University
PO Box 1738
Rotterdam
The Netherlands
jhillegersberg@fbk.eur.nl

Mike Papazoglou
Infolab, Dept. of Info Systems and Info Management
Tilburg University
PO Box 90153
Tilburg
The Netherlands
mikep@kub.nl

Abstract In this paper we present a methodology, called binding Business Applications to LEgacy systems (*BALES*), that allows to blend modern web-services with objectified legacy data and functionality in order to construct new business applications.

The core of this methodology encompasses a linking phase, during which reusable portions of wrapped legacy data and functionality are determined, and subsequently linked to the interface of web-services. In this way, the legacy wrappers are hardwired into the web-service interfaces, and, so they can be reconfigured to meet changed business requirements.

Keywords: web-services, legacy integration, mapping methodology

---

The original version of this chapter was revised: The copyright line was incorrect. This has been corrected. The Erratum to this chapter is available at DOI: 10.1007/978-0-387-35614-3_21
C. Rolland et al. (eds.), *Engineering Information Systems in the Internet Context*

1. INTRODUCTION

Many companies use enterprise information systems that are becoming imperative to conduct business, but at the same time feel hindered by them to swiftly react to new customer demands (Wong, 2001). This is partly due to the fact that the homogeneous perspective on integrated enterprise architectures assumes detailed negotiations to agree upon business protocols, and semantics. The resulting integrated enterprise applications are tightly coupled with many interconnections and brittle application interfaces.

The web-services paradigm is heralded as *the* de-facto technology to implement, disaggregated and loosely coupled EISs on the fly, without virtually any prior-design time cooperation.

Broadly speaking we may characterize a Web-service as a business-centric software component that adheres to Web standards, performs business related activities and may conduct transactions over the Internet. At the most abstract level, web-services refer to the resources accessible via the Web which provide structured data sources (e.g., databases), semi-structured (e.g., HTML documents) or unstructured information sources (e.g., text files, images), and computational software (e.g., business processes/applications, workflows, or agents). The availability of diverse electronic services on the Web has raised high expectations for flexible and efficient sharing of services, which has been witnessed in the areas such as electronic catalogs, digital libraries, Web-based value chain networks, Web-based healthcare systems, just to mention a few. Integrated value chains have significant competitive advantages over traditional enterprises. They can provide new services and products without the investment and delays a traditional enterprise requires, and may utilize the best-in-their-class component services without having to develop them.

Web-services are registered and classified in a central repository system as a collection of web-service descriptions that represent their properties (contact information, service location), capabilities and price, as well as information regarding their semantics, security, and service contents. Potential customers can browse through the available web-services, retrieve them, and combine them into new value-adding enterprise application while actively trading. This can be technically achieved by supporting various flavors of web-service composition, e.g., explorative, semi-fixed and fixed composition (Yang et al., 2002), depending on mechanisms such as dynamic binding and loose coupling. The (bundled) web-services are then invoked and deployed decentrally at the suppliers site, using web-service middleware, e.g., the Simple Object Access Proto-

col (SOAP). Examples of web-services are payment-services, auction-services, ordering-services, and tracking-and-tracing-services.

One critical requirement of constructing new business applications by fusing web-services is their ability to selectively identify reusable and modifiable portions of legacy wrappers and to combine them with new web-services in a consistent manner. Legacy wrappers denote a broadly accepted technique letting new web-services gracefully co-exist with old-legacy systems. This is called the *access/integration in place* strategy.

Legacy systems refer to geriatric enterprise information systems which are hard to adapt to new business requirements. Written in legacy programming languages, legacy systems are still a valuable asset to modern organizations. However, developers face many difficulties in exposing and using legacy functionality in modern development projects. Current modern development processes usually do not provide tools, techniques, or guidelines on how to utilize legacy functionality. The standard assumption seems to be that software components, and more recently, web-services, are developed from scratch.

We foresee that web-services, that quickly gain popularity, will often be developed on top of existing legacy systems in order to expose useful legacy functionality to modern web-based, and possibly inter enterprise systems. To support the development of such web-services, we present a methodology in this paper that facilitates the "vertical" integration of web-services with encapsulated legacy systems and databases. The methodology is labelled BALES: Business Applications to LEgacy systems. BALES provides a modelling "front-end" to existing wrapping technologies.

The remainder of this paper will explain BALES and its phases using a running example. The example is inspired from building block definitions that we helped develop at the Department of Defense in the Netherlands for a depot at an airbase.

2. THE METHODOLOGY

An important underlying assumption of BALES is that web-services usually correspond to a high level business workflow that can be modelled using 'traditional' forward engineering, object-oriented, modeling techniques. The forward engineering phase results in an enterprise model of the web-service containing constructs such as business tasks and entities/objects. In this research, web-services can be reflected by business worklfows that assign business tasks to actors according to the state of each business task in progress, and, move the business process forward from one role (that performs a business task) to the next. Web-services are based on an extensive foundation of business task ob-

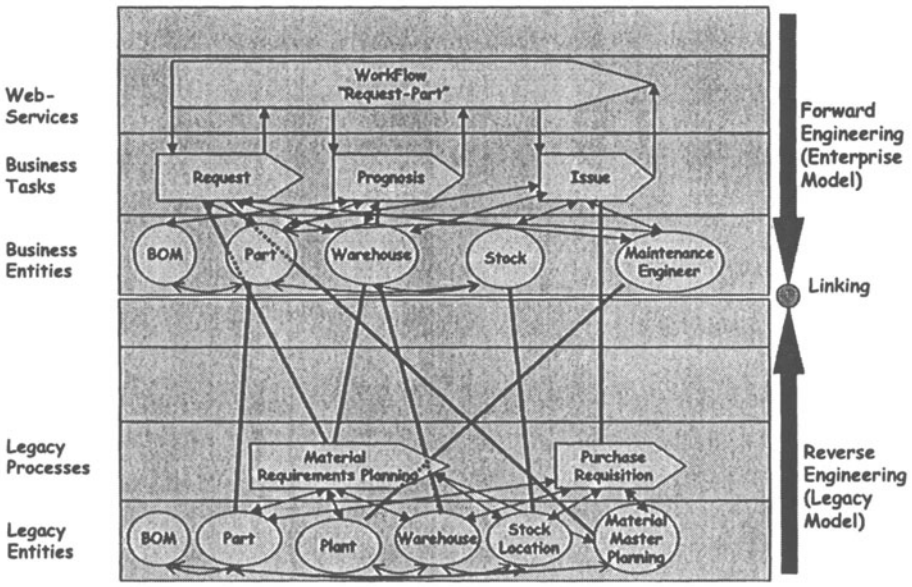

Figure 1. Developing an Enterprise Model by reusing Legacy Wrappers.

jects that form the basis for building new distributed applications. Web-service enabled business processes, e.g., mortgage processing, can track transactions across department and even enterprise boundaries, however, this "horizontal" integration is not any further considered here.

BALES however goes further by combining this forward engineering phase with reverse engineering of the related legacy system(s) present. This results in a legacy model that is linked to the enterprise model. The enterprise model, legacy model and linking are illustrated in Figure 1 using the defense example. The upper part of this figure illustrates the results of the forward engineering of the business domain (phase 1) in terms of workflows, business tasks and business entities.

The lower part of the picture 1, depicts 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 six 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 on 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.

The BALES linking phase makes use of the Component Definition Language (CDL), a superset of the OMG Interface Definition Language (IDL). CDL is used to specify both legacy and business objects in order to facilitate a search for matching objects. In additional advantage of using these already existing specifications, is that for IDL many mappings to traditional 'legacy' languages such as COBOL and C are defined. This makes BALES applicable to reverse engineer any legacy system, as long as it is written in a language that has an IDL mapping.

Another notable characteristic of BALES is the notion of 'parameterized business objects'. These are explicit mappings added to the business objects to connect them to their legacy counterpart. The parameterized business objects are an important result of the application of BALES.

BALES has its own metamodel that includes extensions to the CDL metamodel, and is used to model the BALES repository. The repository has been implemented in ConceptBase. The underlying representation language of ConceptBase is O-Telos. O-Telos is an extension of the Telos language, that has been specifically designed at the beginning of the 90s ((Mylopoulos et al., 1990), (Mylopoulos, 1992)) for supporting software development during all phases of the lifecycle, from analysis down to implementation and maintenance. Telos is a formal knowledge representation language, capable of storing (temporal) facts, e.g., about analysis models, and deductively infering new facts. This language is based on first order logic and has a frame syntax to represent and query classes and objects.

The *BALES* roadmap is basically organized in three main phases to assist the designer when designing parameterized business objects, that are summarized in the following (see Figure-2):

- *Forward Engineering Phase.* The forward engineering phase aims at constructing an (integrated) enterprise model as a collection of collaborating web-services that are made up of a business objects, and modelled according to a stratified architecture. The resulting enterprise model is subsequently transformed into a textual CDL specification and subse-

quently linked to the *BALES* metamodel in order to facilitate the discovery of matching legacy wrapper specifications.

- *Reverse Engineering Phase.* During the reverse engineering phase, the wrapper specifications are connected to the meta model in a similar way. In some cases, a legacy object model might be available that represents legacy databases and applications as a set of cooperating legacy entity and process objects. This model is then translated into a textual description defining all objects and their interconnections using the *BALES* dialect of CDL, and associated to the meta model.

- *Linking Phase.* The goal of the linking phase is to construct parameterized business objects in terms of selectively wrapped partitions of legacy systems. The input of this phase consists of both the instantiated enterprise model and wrapper specifications which result from the forward and reverse engineering phase. Both categories of specifications are thus connected to the meta model and stored in a repository system.

  During this step, enterprise model specifications are firstly matched to wrapper definitions and subsequently linked to reusable partitions of wrappers. The process of identifying equivalent relations between business objects and legacy objects. that are stored in some kind of repository system, is called *matching*. After thorough analysis of the results of the matching activities, the business objects are connected to the appropriate (portions of) legacy objects. This activity is denominated *mapping*. Due to the complexity of these activities, the linking phase is partitioned in four related sub-tasks:

  1 Semantic Content Matching.
  This step is directed towards identifying *semantically* relevant wrapper specifications for a target (enterprise model) specification. In particular, the semantic equivalence between the set of labels for CDL constructs (also referred to as the semantic content) within an enterprise model specification and a collection of available legacy resources is determined.

  2 Type Tree Matching.
  The legacy resource specifications can then further be investigated by comparing the CDL types of both categories of both enterprise model constructs and wrapper definitions. This is achieved by deriving type trees that anatomize CDL specifications to the level of

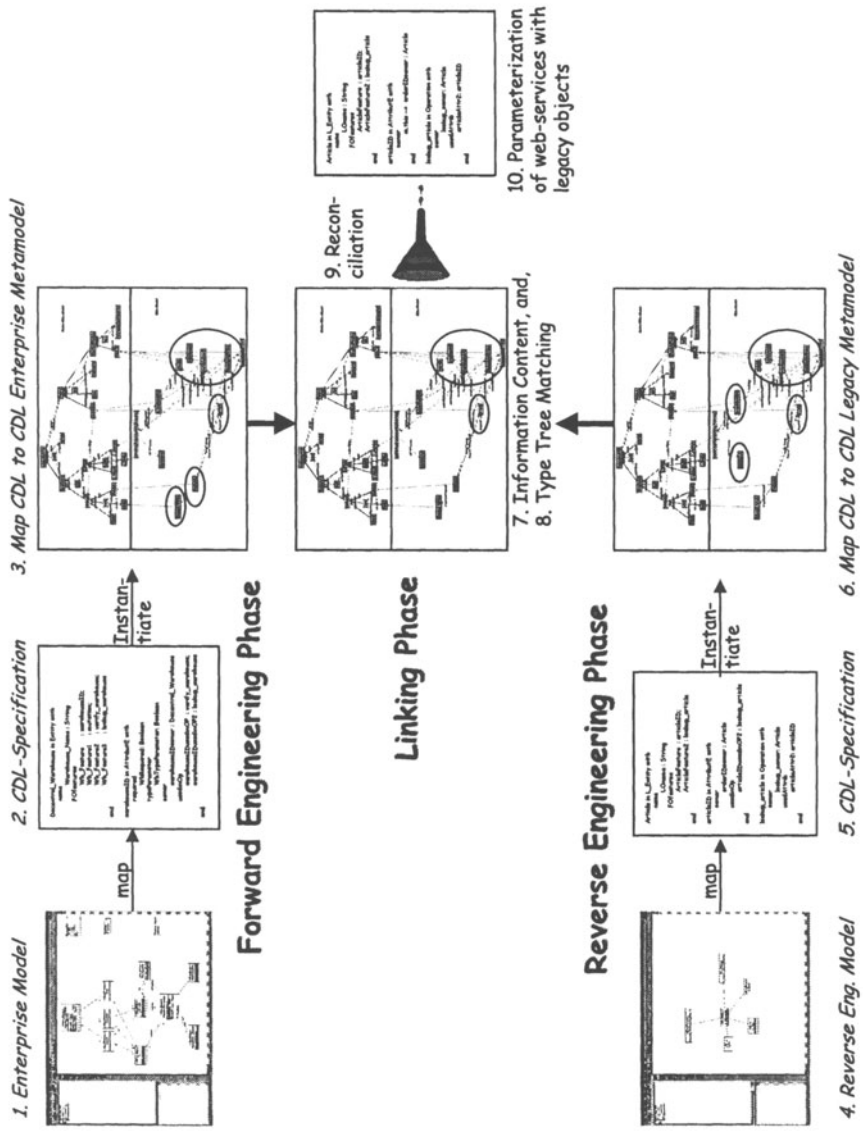

Figure 2. The *BALES* Methodology.

atomic data types, and then calculating the resemblance of type trees of available legacy resources with the type tree of the required business objects.

3 Reconciliation.

The semantic content matching and the type tree matching phase both result in a measurement (coefficient). The former expresses the overlap of the semantic content (labels) of CDL specifications, while the latter computes the overlap of object types used within specifications. During the reconciliation phase, the designer interprets both coefficients and ascertains the strategy for dealing with available legacy resources. In case of considerable semantic overlap of the semantic content of both the enterprise model and the wrapper definitions, semantic mismatches are resolved during this phase as a preparation to the actual parameterization process.

4 Parameterization.

After reusable legacy object candidates have been identified, the designer may decide upon parameterizing (portions of) the business objects. In that case, he needs to further investigate type conformance between two specifications in order to avoid typing conflicts at runtime. This is basically achieved by proving formal substitutability between a business object and legacy wrapper interface definition. Substitutability is an well-investigated technique in the domain of *component composition*. A component, `C`, is said to be a substitute for component, `D`, if component, `C`, can replace the other component, `D`, without somebody noticing it. After substitutability has been formally proven, the designer can parameterize the business object with fragments of legacy wrapper specifications. These links are hard-wired into the business objects so that specific method calls can be marshalled from them to one or more linked legacy resources.

3. FORWARD ENGINEERING THE BUSINESS

The forward engineering phase consists of activities to transforms a conceptual enterprise model into a CDL and to map the resulting CDL model specification to a meta model which serves as a basis for comparison between business and legacy enterprise models. This phase thus comprises the following

activities which correspond to steps 1, 2, and 3 in fig. 2, viz. (1) enterprise modeling, (2) specifying business object (CDL) interfaces on the basis of this model, and, (3) connecting them to the meta-model.

3.1 Step 1: Creating an Enterprise Model

The forward engineering activity starts with the construction of an enterprise model. Enterprise models reveal the activities, structure, information, actors, goals and constraints of the business (Papazoglou and Heuvel, 2000a). These models can be viewed from multiple complementary perspectives to enhance their understanding, e.g., from a business rule view, an activity view (IDEF), a business process view and an organization view (Loucopoulos and Karakostas, 1995). The last years have witnessed the convergence of multi-perspective enterprise modeling and software development, particulary with the advent of object, component and web-service technology. An example of an OO enterprise modeling approach is reported in (Marshall, 2000).

The logical structure of enterprise models can be represented both visually, e.g., by applying UML or textually, e.g., by deploying XMI or WSDL (see: http://www.uddi.org/). WSDL is an interface specification language that has been tailored for specifying the behavior of web-services.

To demonstrate the result of the enterprise modeling activity, we have drafted the enterprise model for the defense case study in terms of the *BALES* UML modeling notation in Figure-3 [1]. The business entities, tasks and workflow have been stereotyped according to the modeling conventions. In addition, more detail with regard to the associations (e.g., multiplicity and role names), and data types of both attributes and operations is incorporated in this model. For reasons of simplicity, we will focus on a subset of the business entities which are reflected in this enterprise model during the remainder of this article.

3.2 Step 2: CDL Specification of the Web-services

Typically, the enterprise model which result from Step-1 of the methodology merely incorporates the essentials of the business domain, abstracting from many "non-relevant" details. For example, often the specification of methods is constrained to their name and possibly some input parameters only. In order to increase the level of detail of the business objects which together reflect the redesigned business processes within the organization, the UML model is firstly transformed into a textual counterpart during step-2 of the methodology. This is needed to be able to compare the business objects in the enterprise

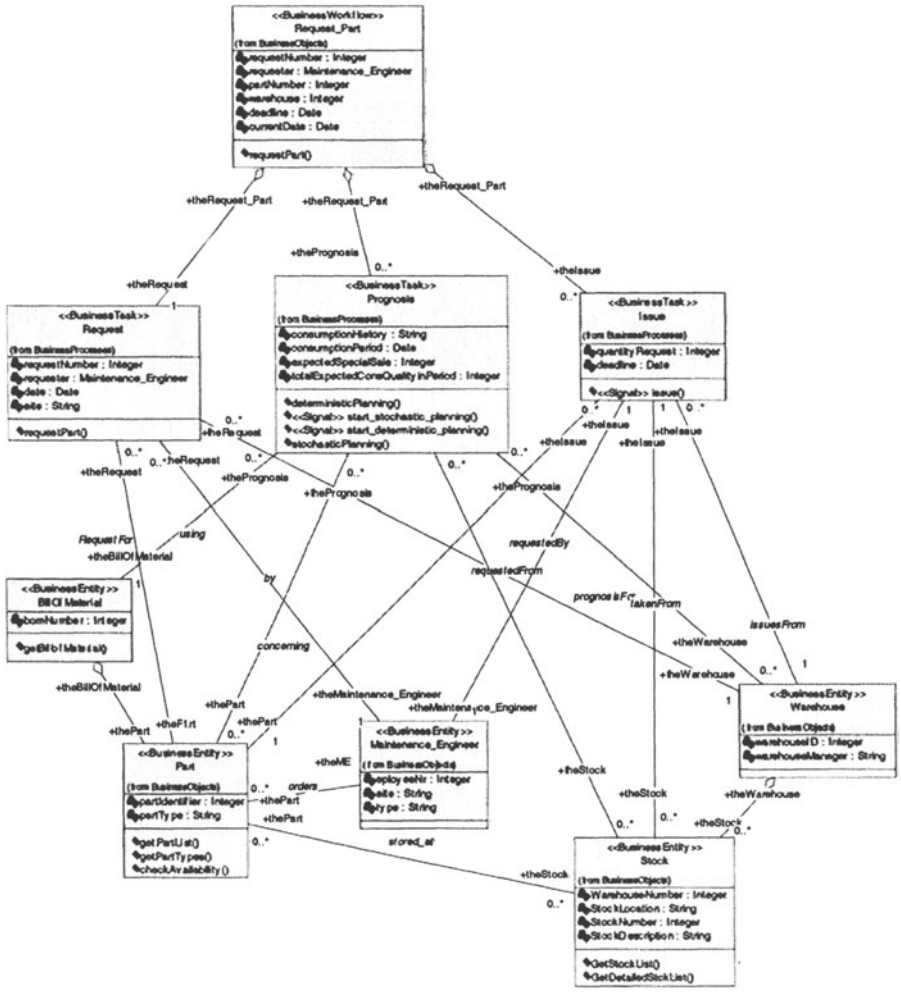

Figure 3. Enterprise Model of Maintenance and Overhaul Business Processes.

model later on with legacy wrappers. The textual specifications are stated in the *BALES*-variant of the Component Definition Language (CDL).

The following excerpt shows the CDL-specification of the `Stock` class, which corresponds to the UML representation in Figure-3:

```
keys={StockNumber}] BusinessEntity Stock {
    [REQUIRED]
    attribute Integer WarehouseNumber;
    [ALWAYS]
    attribute String StockLocation;
    [ALWAYS]
    attribute Integer StockNumber;
    [ALWAYS}
    attribute String StockDescription;
    Integer GetStockList (in Integer StockNumber,
                          in Integer WarehouseNumber);
    void CreateStock (in Integer StockNumber,
                      in Integer WarehouseNumber,
                      in String StockLocation,
                      in String StockDescription);
    }; // End: Stock
```

3.3 Step 3: Instantiating the Metamodel

After the interfaces of both the business objects and processes (hence tasks and workflows) have been specified in CDL, the CDL specifications are instantiated to the Enterprise (CDL) Meta model. This model renders the *instantiations* of the CDL enterprise model components. It thus illustrates how the CDL and model specific constructs are related to each other and conceptualizes information about their types. The CDL instantiation step is used as basis to infer how the constructs found in a Enterprise Meta Model can be connected to (viz. re-use) related constructs found on the Legacy Meta model.

Figure 4 shows an excerpt of the abstract representation of enterprise model displayed by the ConceptBase graph browser, that corresponds to the CDL specification presented in the above. The upper half shows the CDL-Metamodel, i.e., the meta classes to represent CDL constructs. Note the distinction between business object types (*BusinessType*) and legacy object types (*LegacyType*). The lower half displays an excerpt of the result of the instantiations of CDL representations to the Enterprise Meta-CDL model for the business entity `Stock` that is instantiated from the as a `BusinessEntity`. The snapshot illustrates three attributes, two of which, `StockNumber` and `StockLocation` serve as an input parameter for the business operation `GetStockList`.

The following excerpt illustrates a small excerpt of the textual Telos frames counterpart of Figure-4.

```
Stock in BusinessEntity with
    TypeKey
          stockkey : StockNumber
        FOfeatures
                StockFeature  : WarehouseNumber;
        StockFeature1 : StockLocation;
                StockFeature2 : StockNumber;
        StockFeature3 : StockDescription;
            StockFeature4 : GetStockList;
        StockFeature5 : CreateStock
end

WarehouseNumber in AttributE with
        typeParameter
          wnumType : CORBA_integer
        required
          wnumReq : YES
end


....

GetStockList in CORBA_OperationDef with
  ownsreturnType
    returnTypGetStockList : CORBA_integer
  ownsinputparSpec
    GSLhasParam  : StockNumber
  ownsinputparSpec
    GSLhasParam2 : StockLocation
end


StockNumber in InputParameter with
  withDirection
    dir : In
  parameter_of
    pof : GetStockList
  withtype
    acnrtype : CORBA_integer
end
```

Specifically, the excerpt in the above demonstrates how the meta-class `BusinessEntity` is instantiated by the entity `Stock`. After the meta-class `BusinessEntity` has been instantiated, Telos objects are created for defining its attributes and operations by instantiating the meta-classes `AttributE` and `CORBA_OperationDef`, e.g., the tokens `StockNumber`, and `WarehouseNumber`, `StockLocation`, `StockNumber`, and `GetStockList` are instantiations of subsequently the attributes `TypeKey` and `FOFeatures`.

In summary, the CDL Meta model serves as an 'independent' canonical model to which the forward as well as the reverse engineered CDL models can be linked, superimposed, and compared in order to ascertain which (portions of) legacy processes and objects can be reused at the enterprise model level.

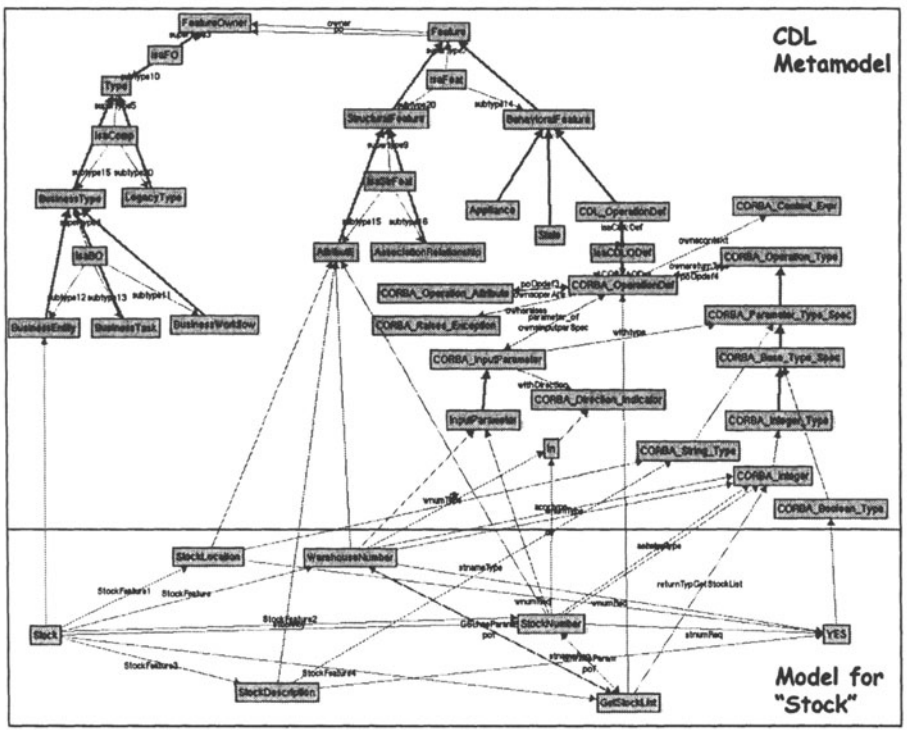

Figure 4. A snapshot of ConceptBase graph browser containing the Enterprise Model.

4. REVERSE ENGINEERING LEGACY SYSTEMS

In the second main phase of the *BALES*-methodology, the legacy entities and processes are represented in terms of CDL and linked to a Legacy CDL Metamodel. The activities during the reverse-engineering phase are largely similar to those performed during the forward-engineering phase. The following three activities, corresponding to steps 4, 5 and 6 in fig. 2, are completed: (4) representing legacy wrappers visually in a legacy model, (5) specifying the legacy wrapper and, (6) instantiating the CDL Legacy Metamodel. Due to reasons of brevity, a description of Step-4 has been omitted.

4.1 Step 5: Specifying the Legacy Wrapper

The following excerpt shows a *wrapper* specification that encapsulates the CDL specification of the legacy entity object `StockLocation`:

```
keys={StockIdentifier}] LegacyEntity StockLocation {
     [REQUIRED]
     attribute String StockCoordinates;
     [ALWAYS]
     attribute Long StockIdentifier;
     [ALWAYS]
     attribute String StockName;
     Integer getStockLocation (in Long StockIdentifier,
                               in Integer StockCoordinates);
     void createStockLocation (in Long StockIdentifier,
           in Integer StockCoordinates, in String StockName);
     }; // End: StockLocation
```

This `LegacyEntity` wraps a relational database table that stores information regarding stock locations with some (mostly free) capacity. In particular, this CDL code designates an encapsulated relational table with transactions which might serve as a candidate for supporting the business entity *Stock*. The number of parts at a given storage pile, labeled with the identifier `StockIdentifier` within a certain location and referred to by the string `StockCoordinates` (this variable name stems back from the time the enterprise had only one stock location with two warehouses), is calculated with the legacy operation `getStockLocation`.

4.2 Step 6: Instantiating the CDL Legacy Metamodel

After the CDL-descriptions of the legacy components are available the legacy CDL specifications are instantiated to a CDL Legacy metamodel much in the same way that we described for the enterprise model.

Figure 5 depicts an excerpt of the abstract representation of legacy object model displayed by the ConceptBase graph browser, that corresponds to the

CDL specification presented in the above. The upper half shows the CDL-Metamodel, i.e. the meta classes to represent CDL constructs. Note the legacy object types (*LegacyType*), `LegacyEntity` and `LegacyProcess`. The lower half displays an excerpt of the result of the instantiations of CDL representations to the Legacy CDL metamodel for the legacy wrapper `StockLocation` that is instantiated from the as a `LegacyEnt`.

```
StockLocation in LegacyEntity with
    TypeKey
          stockkey : StockIdentifier
        FOfeatures
        StockFeature1 : StockCoordinates;
        StockFeature2 : StockIdentifier;
        StockFeature3 : StockName;
        StockFeature4 : GetStockLocation;
        StockFeature5 : CreateStockLocation
end

StockCoordinates in AttributE with
  typeParameter
    wnumType : CORBA_String_Type
  required
    wnumReq : YES
end

...

GetStockLocation in CORBA_OperationDef with
  ownsreturnType
    returnTypGetStockList : CORBA_integer
  ownsinputparSpec
    GSLhasParam  : StockIdentifier
  ownsinputparSpec
    GSLhasParam2 : StockCoordinates
end


StockIdentifier in InputParameter with
  withDirection
    dir : In
  parameter_of
    pof : GetStockLocation
  withtype
    acnrtype : CORBA_long
end

...
```

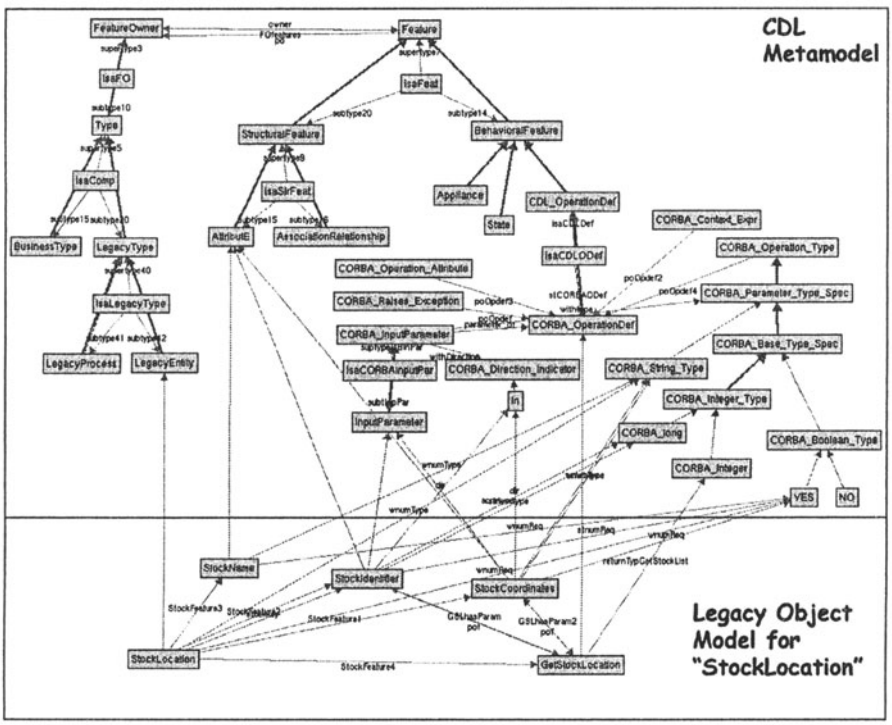

Figure 5. A snapshot of ConceptBase graph browser containing the Legacy Wrappers

5. MAPPING BUSINESS OBJECTS TO LEGACY WRAPPERS

After the CDL descriptions of the forward and reverse engineered models have been instantiated to their respective CDL meta-models, the two models can be superimposed and compared (see Figure- 2). The following four tasks need to be completed in succession to link legacy objects to business objects: semantic content matching (step 7), type tree matching (step 8), reconciliation (step 9), and finally parameterization (step 10). The first two activities distill those legacy wrapper specifications from the repository system which are relevant from a semantic content and type conformance perspective. The first two steps thus aim at limiting the space of legacy wrappers that need to be considered as legacy candidate sources for business objects. For reasons of brevity, we will not go into further detail here, but instead take a closer look at step 9 and 10.

5.1 Step 9: Reconciliation

Both the semantic content matching and the type tree matching phase, resp. step 7 and 8 in our approach, yield in metrics to express successively the ontological proximity and type tree resemblance of an target enterprise modeling CDL specification with available legacy wrapper CDL resources. These metrics now need to be interpreted by the designer, and, whenever both measurements are satisfactory, the legacy wrappers can be semantically enriched as a preparation to the parameterization.

In principle, four alternative strategies can be defined to deal with the available legacy resources, depending on the outcome of the coefficients for (information) content semantics and object model semantics (see Figure-6):

1 Ignore Wrapped Legacy System (modules).

This strategy is applicable in the case that both the metrics indicating the ontological relevance and similarity of type trees are low. In the running example, a low value of both semantic and schematic object similarity is defined below the threshold values of .70. Existing legacy resources

are retired, and the new business application is required to be constructed *from scratch.*

2 Adapt the Wrapped Legacy Systems.

This scenario is feasible when the CDL specification from the enterprise model and one or more legacy wrapper CDL definitions match relatively well at both the semantic and schematic level. In this case, the designer is allowed decide to change the legacy wrapper functionality, e.g., by streamlining the terminology or complementing the wrapper with additional required business logic. (Semi-)Automatic adaption of component signatures has been investigated in (Reussner, 2001).

3 Change the Enterprise Model.

Another strategy that is suitable whenever business and legacy objects are relatively similar at both the content semantic and object model level, is to adapt the enterprise model so that it better fits to available wrapper definitions.

This "upstream" strategy is most likely not desirable in many cases, because it would imply that only minimal improvements can be implemented to the existing business processes.

4 Reuse the wrapped legacy systems (Configure).

This scenario represents the ideal solution. This strategy is most applicable in case of a close, or even a full, match between business objects that are laid down in the enterprise model(s) and the wrapped legacy system (modules).

In reality, a mixture of these strategies will often be applicable and different parts of the enterprise model exhibit different measurements of similarity with available legacy resources. Empirical experiments based on a data set derived from real-world projects are required to infer and calibrate intervals for both the metrics of semantic content matching and the type tree matching metrics, so these strategies can be positioned in a more meaningful way. These experiments could for example be set up to observe the impact of applying one of the four strategies within certain value ranges of both measurements to the return of investment of companies.

Both the semantic similarity measurement and the type tree matching coefficient are larger than the threshold values, T_{SimSem} and T_{SimOms}, thus resulting in the conclusion that parameterization is the most efficient strategy here. The

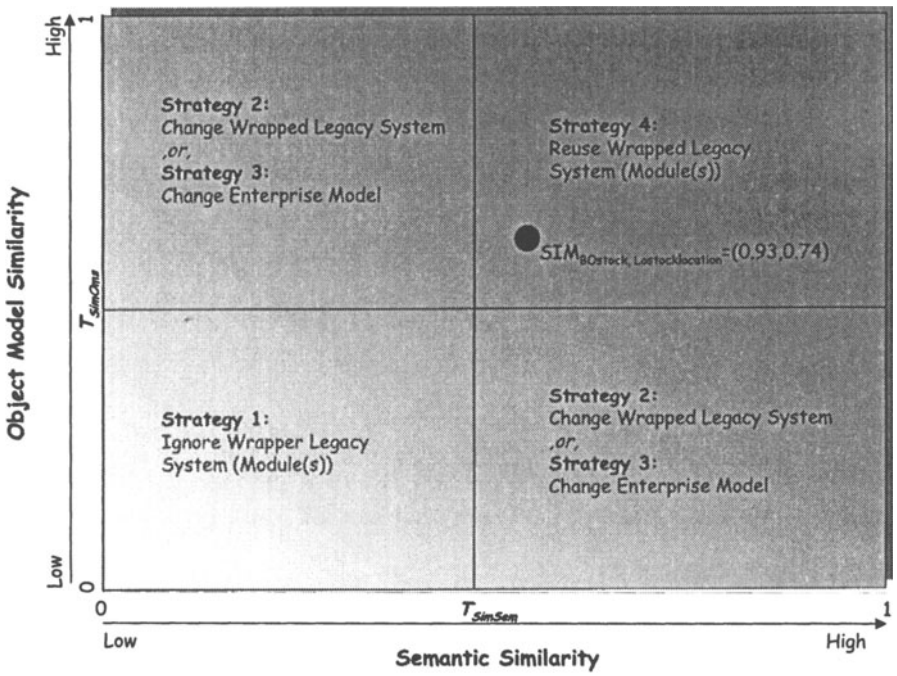

Figure 6. Scenarios for dealing with Wrapped Legacy Systems.

details of determining both coefficients is not included here, due to reasons of space. We refer to (heuvel, 2002) for an in depth discussion of both activities.

5.2 Step 10: Parameterizing business objects with Legacy Objects

During the last step of the *BALES* approach, appropriate legacy wrapper definitions are hard-wired in the CDL business object specifications obtained in step 2 of the forward engineering phase. To achieve this, first the designer has to ascertain that business object interface can be substituted by (portions of) a legacy wrapper definition. A strong requirement is that the substitutability needs to be proven formally up till the parameter type relationship (co-variant result types and contra-variant input parameter type(s)). A complicating factor, is the fact that in many cases substitution can be achieved by manipulating the legacy object interface logically or mathematically.

For the purpose of parameterization, we have extended the CDL language with a special linking operator 'X ? ! Y'. The variable `X` denotes a source construct from the *BALES* enterprise metamodel, e.g., the attribute `StockNumber`, and `Y` denotes the target construct from the *BALES* legacy metamodel, e.g., the legacy attribute`StockIdentifier`.

Hence, `Y` designates the target construct (e.g., a CDL legacy *Operation*) which corresponds to a reused operation from the Meta-CDL legacy model. Alternatively, a reference to a legacy construct can be included into a business type specification as a parameter, e.g., as an additional operation input parameter.

The resulting business entity specification `Stock` can now be defined as follows:

```
keys={StockNumber}] BusinessEntity Stock {
  [ALWAYS]
  attribute Integer StockNumber->
   LegacyEntity.StockLocation.attribute.StockIdentifier;
  [REQUIRED]
  attribute Integer WarehouseNumber ->
   TRANSFORM
      (LegacyEntity.StockLocation.attribute.StockCoordinates);
  [ALWAYS]
  attribute String StockLocation ->
   LegacyEntity.StockLocation.attribute.StockCoordinates;
  [ALWAYS]
  attribute String StockDescription;
  Integer getStockList ->
   GetStockLocation(in Integer StockNumber, in Integer
```

```
 WarehouseNumber);
void createStock --> CreateStock (in Integer StockNumber,
 in Integer WarehouseNumber, in String StockLocation,
 in String StockDescription);
}; // End: Stock
```

The auxiliary function `TRANSFORM` which is included in the mapping definition of the attribute `WarehouseNumber` translates the attribute `LegacyEntity.StockLocation.attribute.StockCoordinates`into an identifier (of type Integer) that refers to a particular warehouse. This can be simply implemented with an internal mapping table that contains all mapping information.

6. CONCLUSIONS

Most of the existing approaches to develop legacy wrappers are designed around the idea that data and logic residing in a variety of legacy database systems and applications reflects a collection of business objects that represent the business processes within a (virtual) enterprise. They assume that by objectifying (wrapping) and combining these entities in a coherent manner with legacy functionality, legacy systems can be readily used in place.

However, this main assumption is not very realistic given the fact that business logic of modernized business processes is in many cases not any longer faithfully reflected by the corresponding legacy wrappers. It seems more credible that only *portions* of legacy wrappers can be reused by the redesigned business application. Moreover, it is very unlikely that this solution (thus wrapping monolithic legacy applications) can subtly deal with various levels of granularity of newly designed business processes. As a consequence, a business change might result into non-proportional changes at the (legacy) business application level. This concept is referred to as the multiplier effect (Holland, 1995). And thirdly, and possibly most importantly, no *methodology* exits to selectively link legacy wrappers to web-services on the basis of business-driven requirements. In addition, emerging approaches for developing web-services, e.g., modeling approaches in ebXML, are constructed on the assumption that they can be developed from scratch, while neglecting available legacy assets. We firmly believe that this is a severe shortcoming of these approaches, and therefore would propose a more balanced "symmetric" methodology that combines both forward engineering of web-services, and reverse engineering of legacy systems.

The *BALES* methodology which has been presented in this paper takes into account these considerations and aims at designing parameterized business objects, which make up a web-service, so that part of their implementation is supplied by legacy objects. Both the reinvented web-services and the existing legacy objects are organized according to an (integrated) enterprise architecture that reflects modernized business requirements. This approach allows discrete recursion of interface specifications of both the web-services' constituents and legacy objects, and as a result, matching them at various levels of granularity. In this way, it becomes possible to *selectively* reuse portions of wrapped legacy data and functionality and *parameterize* the interfaces of web-services in terms of (parts of) legacy object interfaces.

Notes

1. Pls. note that this simplified enterprise model only depicts the structural relationship between business entities. No dynamic models of the business processes are incorporated.

References

Fingar, P. (2000). Component-based frameworks for e-commerce. *Communications of the ACM*, 43(10):61–66.

Yang, J. and Papazoglou, M. (2000a). Ineroperation support for electronic business. *Communications of the ACM*, 43(6):39–47.

Yang, J., Papazoglou, M., and Heuvel, W. V. D. (2002). Tackling challenges of service composition in e-marketplaces. In *Proceedings of RIDE2002 (to Appear)*. IEEE.

Heuvel, W.J. van den. *Integrating Modern Business Applications with Objectified Legacy Systems*. PhD Thesis, Tilburg University, The Netherlands

Holland, J. H. (1995). *Hidden Order : How Adaptation Builds Complexity*. Reading, Mass., Addison-Wesley.

Loucopoulos, P. and Karakostas, V. (1995). *System Requirements Engineering*. McGraw-Hill Book Comp., London.

Marshall, C. (2000). *Enterprise Modeling with UML: Designing Successful Software Through Business Analysis*. Addison-Wesley, Reading, MA.

Mylopoulos, J. (1992). Conceptual modeling and telos. In Loucopoulos, P. and Zicari, R., editors, *Conceptual Modeling, Databases and Case: An Integrated View on Information Systems Development*. J. Wiley, New York.

Mylopoulos, J., Borgida, A., Jarke, M., and Koubarakis, M. (1990). Telos: Representing knowledge about information systems. *ACM Transactions on Information Systems*, 8(4):325–362.

Papazoglou, M. and Heuvel, W. V. D. (2000a). Configurable business objects for building evolving enterprise models and applications. In Aalst, W. V. d., Desel, J., and Oberweis, A., editors, *Business Process Management: Models, Techniques and Empirical Studies*, pages 328–344. Springer.

Reussner, R. (2001). The use of parameterised contracts for architecting systems with software components. In *Proceedings of the Sixth International Workshop on Component-Oriented Programming (WCOP'01)*.

Wong, L. (2001). E-services: A key component for success. *EAI Journal*, pages 18–25.