



**Application Development Using Codiscent  
Generative Technology and Methodology**  
A White paper

An Overview of Codiscent's GES Platform and Agile  
Generative Engineering Methodology

# Application Development Using CodiScent Generative Technology and Methodology

## Contents

Introduction .....	3
Overview of CodiScent’s Tools and Methodology .....	5
Components.....	5
Methodology .....	7
CodiScent Use Cases .....	9
Database Compare Utility .....	9
Dojo Browser-based Database Interface .....	14
Enhancement of an Interface to Incorporate Temporal Features.....	22
Browser-Editor for COBOL File Data .....	25
Summary.....	28
Coding Leverage.....	28
Reuse .....	28
Architectural Focus .....	28
Rolling Refactoring.....	28
Reduced Maintenance Effort .....	28
Staffing Leverage .....	29
Migration and Transformation.....	29
Working With CodiScent.....	30
Contact CodiScent.....	30

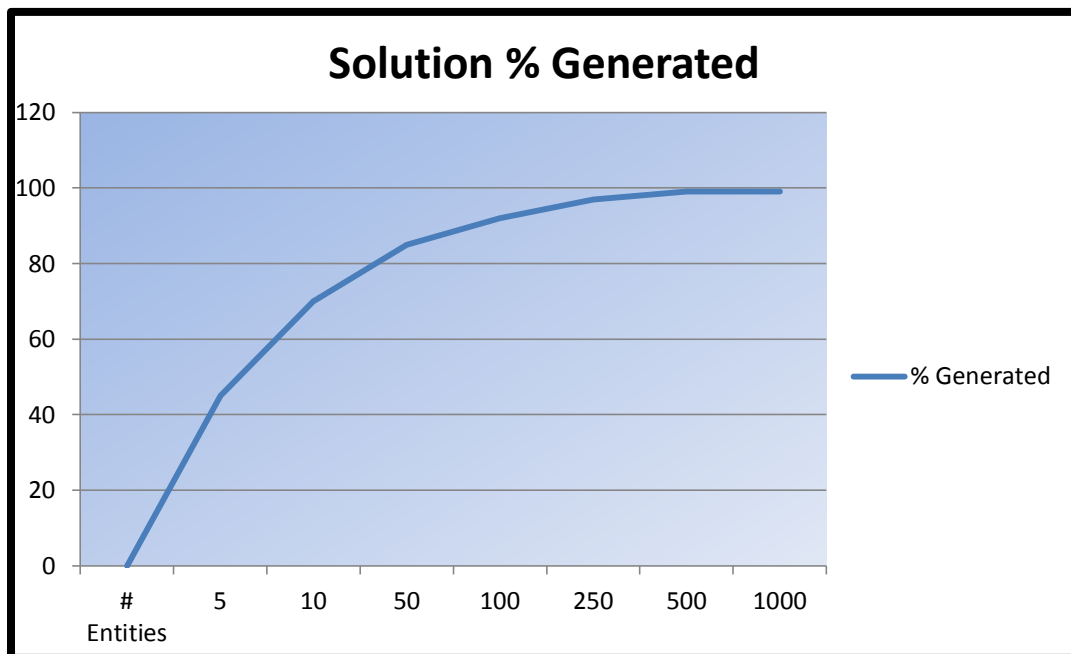
## Introduction

CodiScent Ltd., a consulting and software development company, has developed proprietary technology and a complimentary methodology that enable it to deliver business solutions **Better, Cheaper** and **Faster** than alternative traditional methods. This document expands on the material presented in the previous whitepaper in which CodiScent's development services were introduced. It focuses on *how* CodiScent's tools and methodology can best be employed to achieve the results described in the previous paper. This will be illustrated in the context of three example solutions developed with the CodiScent toolset.

The realization of **Better, Cheaper** and **Faster** software development results from several elements of the CodiScent development tools and methodology:

- **Coding Leverage**—The generated components of CodiScent solutions may consist of anywhere from 75% to approaching 99% generated code. Generally, the larger the domain of the problem being solved, the higher the ratio of generated code will be. Reducing the absolute amount of code written makes it easier to produce defect-free code (Better), the amount of time necessary to produce it (Faster) and the cost of producing it (Cheaper.)

The graph, below, illustrates a representative relationship between the size and scope of the specification set, with number of entities represented as a proxy, and the percent of the application that is generated in the solution:



- **Reuse**—Another advantage of CodiScint solutions is that it is easy to reuse artifacts, which contributes to the return on the investment in developing them. Note in the Database Compare Utility example that the entire solution is reusable. After replacing the original specification data with a new set of metadata drawn directly from the database system tables, a new, error-free solution instance is generated in mere moments (Better, Cheaper and Faster.) Reusing the existing solution cut costs and time to deliver while simultaneously eliminating development risk.
- **Architectural focus**— CodiScint’s technology and methodology lend themselves to focus on defining solution architectures independent of implementing them and results in enforcement of best practices throughout the code base. This produces greater separation of concerns among components and more abstract and flexible components that are easier to repurpose and reuse (Better.)
- **Rolling refactoring**—A common outcome of repeated iteration in the course of development using standard practices is accumulation of code that should be refactored but which isn’t due to expedience. Given the leverage and rapidity with which CodiScint solutions are built, refactoring takes place over the course of development as a natural result of iterative refinement. This results in cleaner, more efficient, more reusable code (Better) and the attendant cost benefits that come with it (Cheaper.)

The advantages, cited above are primary benefits achievable with CodiScint. In addition, there are second-level benefits to which the primary benefits contribute:

- **Reduced Maintenance Effort**—Reusability and Architectural Focus both contribute substantially to the maintainability of CodiScint solutions. This enhances ROI significantly by reducing the Total Cost of Ownership (TCO) of the solution, which includes acquisition, operational and maintenance costs over its usable life.
- **Staffing Leverage**—The Architectural Focus and Reusability of CodiScint artifacts facilitate optimal programming resource allocation in two ways: first, it allows different components of a solution to be developed independently of one-another, possibly by programmers that are expert in specific elements of the implementation technology and, secondly, it allows the most experienced and capable programmers to develop core solution components that can then be disseminated and incorporated in multiple projects. Having a library of reusable assets can result in improved organizational agility in addition to all of the other benefits of CodiScint technology.
- **Migration and Transformation**—Separation of concerns is a significant design goal for any software system. Support for isolation and encapsulation, which is inherent in the CodiScint approach, lends itself to architecting solutions that can be transformed to accommodate new infrastructures or provide modified functionality with a minimum of revision.

The remainder of this whitepaper contains three sections:

- An overview of CodiScint’s tools and methodology
- Development use-cases:
  - Database Compare Utility
  - Browser-based Database Interface using Dojo components and Derby or Oracle Databases

- Enhancement of the previous use-case to incorporate temporal features that enable the user to visualize the data as it looked at any time in the past or will look in the future
- COBOL text selective parser for processing COBOL copybooks.
- A summary and analysis of time and cost to implement these solutions, stressing the benefits and ROI of CodiScent tools, as we have identified them, above.

## Overview of CodiScent's Tools and Methodology

### Components

CodiScent's tools include the Projector Template Generator (**PTG**), the Generative Engineering Studio (**GES**), the Relational Metadata Inference Transformer (**RMIT**) and the Configurable Graphical Interface Factory (**CGIF**).

The **Projector Template Generator (PTG)** is the heart of CodiScent's delivery system. It employs clear, intuitive and exceptionally flexible templates which, when linked to specifications (metadata that describe solution requirements) can generate nearly anything in any format—code, data or text. PTG output is independent of the rest of the CodiScent platform and can be further developed without using it; however, there are significant benefits to continuing to use the platform throughout the developed software's life.

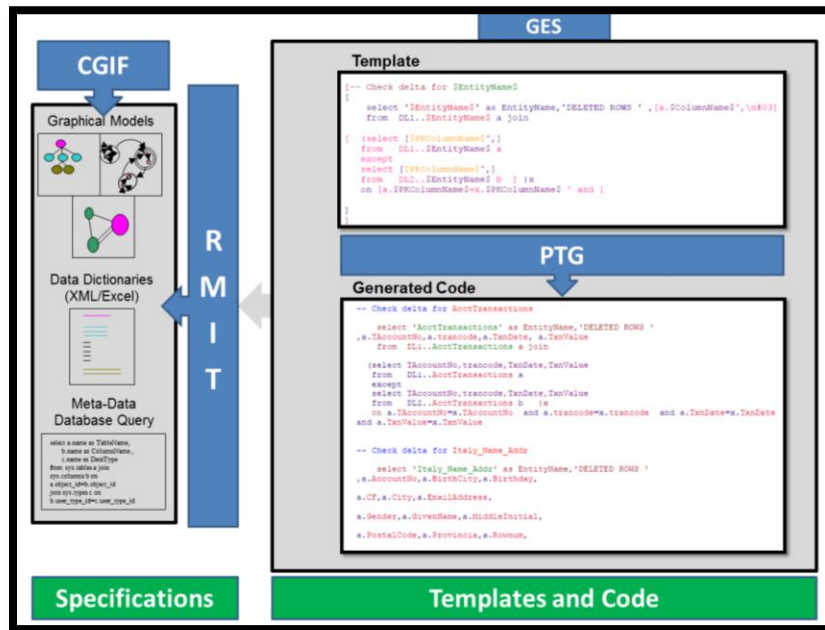
The **Generative Engineering Studio (GES)** is an IDE that facilitates building and managing the assets—specifications, templates and generated code—associated with generative development projects. GES works with many types of specifications from textual (XML/Excel/Text/SQL Result Sets) to diagrams or graphical models and can easily interface with third party modeling and metadata repositories, as well. Working with GES is designed to be completely consistent with CodiScent's methodology and can reduce development costs by as much as 60% or more as compared with alternative software development methods. This results in high-quality software, delivered at very competitive cost and in short time frames.

The GES employs these two components to link to and map specification data:

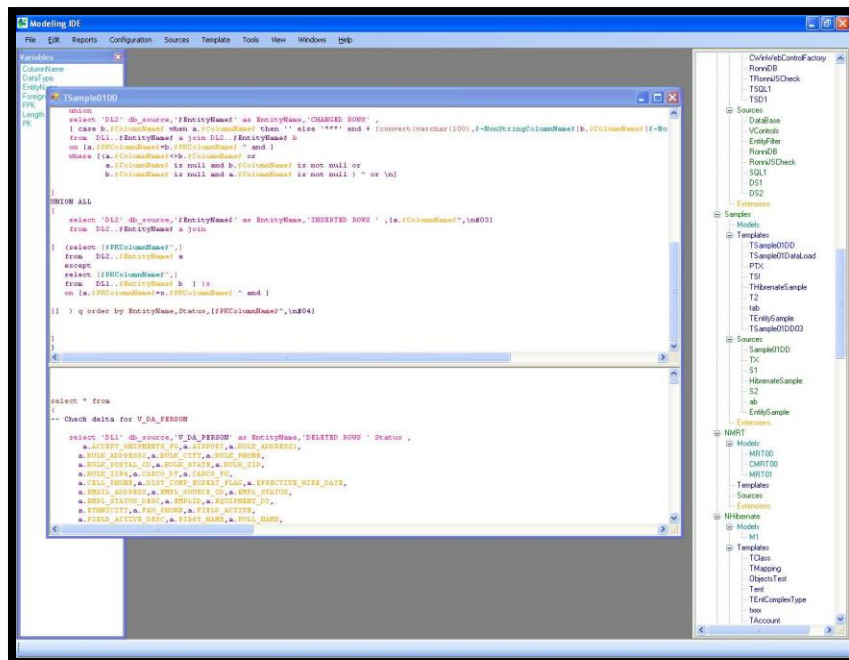
The **Relational Metadata Inference Transformer (RMIT)** is a graphical tool that enables GES users to define a cohesive relational map for heterogeneous specification data structures so that they can be used to drive generated output through PTG templates. The ability to coalesce multiple data source types provides exceptional flexibility to build detailed and nuanced models of the problem domain to which the CodiScent toolset is being applied. The RMIT also provides flexible analytical support for viewing and ensuring metadata consistency prior to the code generation process.

The **Configurable Graphical Interface Factory (CGIF)** is a tool that enables a GES user to define customized diagramming schemes and link them to specification data structures that can be accessed through the RMIT and used to drive generation through the GES. This allows a user to employ graphical models where they are clearer and easier to use than relational or XML data, for instance. Using the CGIF meta-interface, the designer can define semantics including shapes, shape relationships (containment/association) and the attributes with which each shape is associated.

The schematic, below, shows the relationships among the components:



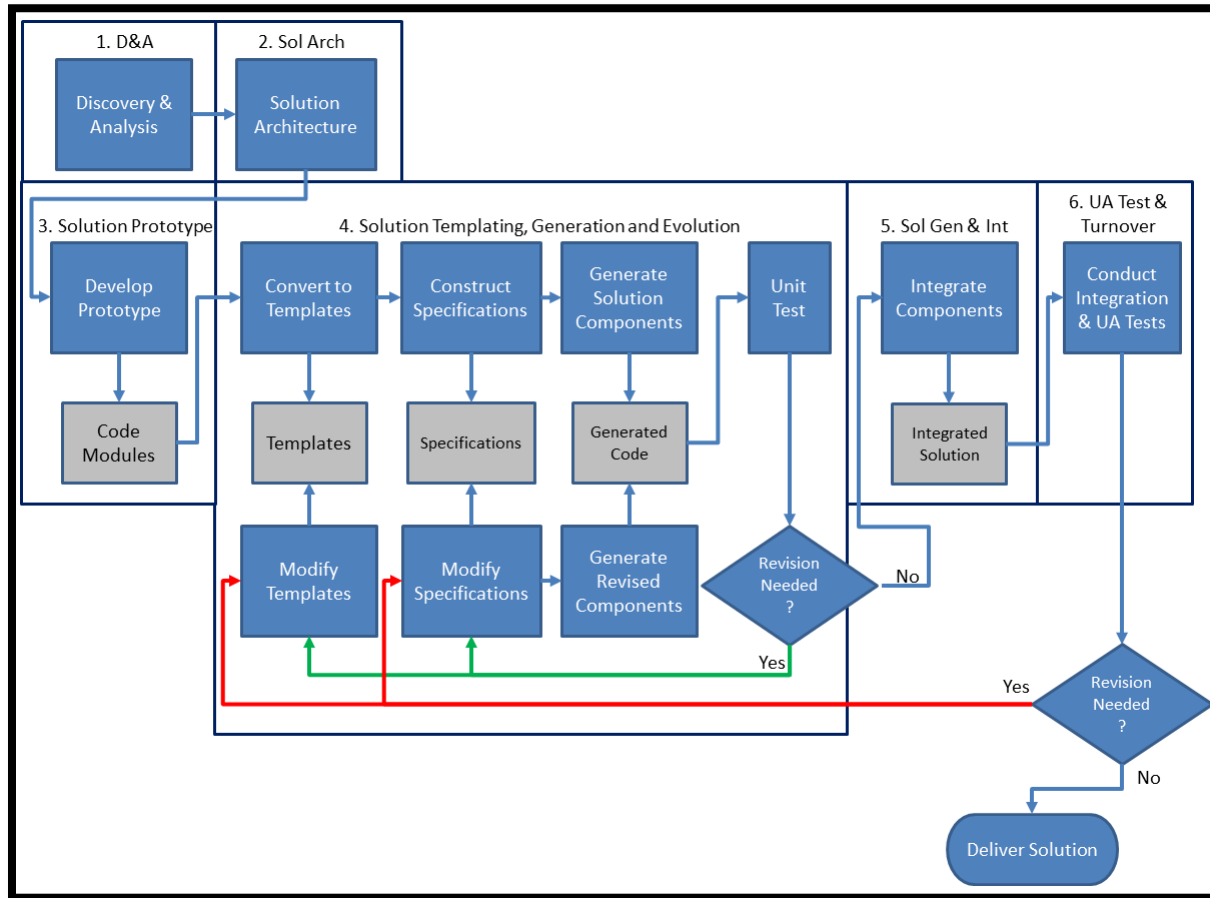
Below, is a screen shot of the GES IDE. The split-screen window contains a template on the top and the code generated from it beneath. Changes in the generated code resulting from modifying the template can be viewed in near-real time. Behind the popup window are configurable panels that provide access to object trees for artifacts in the GES repository, such as specifications sources, graphical depictions and templates.



## Methodology

CodiScen's Agile Generative Engineering Methodology (**CAGEM**) combines acknowledged best-practice project management practices with an agile lifecycle approach that balances rigorous planning and control with agile, evolutionary solution design and implementation.

The flowchart, below, portrays CodiScen's phased delivery approach:



CodiScen projects are conducted in the following phases:

1. **Discovery and Analysis Phase:** As with nearly any development methodology, the initial phase focuses on assessing business needs and defining a solution that best fulfills the requirements. Within CodiScen's methodology, however, there is also a focus on identifying opportunities to employ the generation tools to their maximum benefit. Specifically, work processes and solution elements that repeat themselves within the problem domain are noted and evaluated as candidates for generation.

Tasks performed in phase include producing a contextual overview of the problem domain, assessing and documenting candidate requirements, performing a complexity evaluation, identifying dependencies and estimating incremental benefits expected from each of the solution's major capabilities.

Overall, this phase is designed to

- a. identify all candidate functional and non-functional solution requirements,
- b. assess the cost/benefit for each and select the function set to be included in the solution,
- c. define a preliminary solution architecture and produce a component inventory that will provide the selected functionality,
- d. understand interdependencies among the components,
- e. produce baseline scope, time and cost estimates and
- f. define a plan for mitigating implementation risks.

This project phase results in artifacts that can be consistent with any requirements modeling tools that may be in use.

2. **Solution Architecture Phase:** with a proposed architecture in mind, the solution is decomposed into the components that will be required to build it. A strategy for creating each component—generate, hand-build or purchase and integrate—is identified.
3. **Solution Prototyping**—An initial implementation of a minimal but representative set of functionality is developed. In this step, traditional programming techniques are applied to create working prototypes of each of the solution’s major components.
4. **Templating, generation and iterative evolution**—This step is the one in which much of the leverage that creates software Better, Cheaper and Faster is applied. In this step:
  - a. relevant parts of the code implemented for the prototype are translated (refactored) into templates,
  - b. specifications are created which may incorporate database metadata and, potentially, enrichment support data that describes the required solution in the context of the planned architecture,
  - c. components are generated from the specifications and templates and then integrated into a working solution,
  - d. the integrated solution is tested and revisions required to modify behavior or enhance performance are noted,
  - e. changes to the specifications and templates are made and the solution components are re-generated and re-tested and
  - f. this process is iterated until the integrated solution components meet functionality, standardization and performance requirements.
5. **Solution generation and integration**—Once the specifications and templates are finalized the complete set of solution components are generated, other components are integrated and the full-breadth solution is integration-tested.
6. **User Acceptance, Deployment and Turnover** – The solution is subjected to user acceptance review and then deployed and turned over to the user organization.



## CodiScent Use Cases

### Database Compare Utility

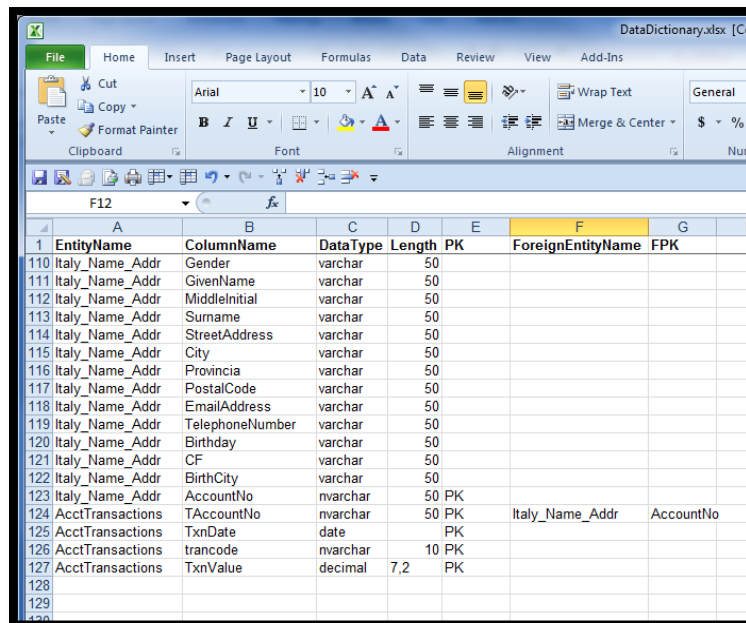
**Use Case:** Produce a utility to compare the contents of two databases that contain identical structures to identify differences between them. This utility is of value to users who need to assess the impact of executing application functions or ETL processes where the before and after states are represented in separate table, schema or database instances.

**Functionality:** Compare the data contained in two database instances and identify added, deleted and changed records. For changed records, identify and highlight the columns in which values have been changed. The output is an SQL result set, with the before and after records positioned one below the other and a marker ('\*\*\*') pre-pended to the changed data value.

**Metadata:** The metadata for this case consists of the SQL Server metadata extracted from the system tables. No enrichment data was required for this case; however, had there not been primary and foreign key constraints in the data, these could have been added to the specifications and imposed exogenously.

**Technology:** This solution was implemented as SQL scripts to be executed through the MS SQL Server management studio.

Here is a sample of the DB metadata in a tabular format:



The screenshot shows an Excel spreadsheet with the following data:

	A	B	C	D	E	F	G
1	EntityName	ColumnName	Data Type	Length	PK	ForeignEntityName	FPK
110	Italy_Name_Addr	Gender	varchar	50			
111	Italy_Name_Addr	GivenName	varchar	50			
112	Italy_Name_Addr	MiddleInitial	varchar	50			
113	Italy_Name_Addr	Surname	varchar	50			
114	Italy_Name_Addr	StreetAddress	varchar	50			
115	Italy_Name_Addr	City	varchar	50			
116	Italy_Name_Addr	Provincia	varchar	50			
117	Italy_Name_Addr	PostalCode	varchar	50			
118	Italy_Name_Addr	EmailAddress	varchar	50			
119	Italy_Name_Addr	TelephoneNumber	varchar	50			
120	Italy_Name_Addr	BirthDay	varchar	50			
121	Italy_Name_Addr	CF	varchar	50			
122	Italy_Name_Addr	BirthCity	varchar	50			
123	Italy_Name_Addr	AccountNo	nvarchar	50	PK		
124	AcctTransactions	TAccountNo	nvarchar	50	PK	Italy_Name_Addr	AccountNo
125	AcctTransactions	TxnDate	date		PK		
126	AcctTransactions	trancode	nvarchar	10	PK		
127	AcctTransactions	TxnValue	decimal	7,2	PK		
128							
129							
130							

The metadata describes data structures for Customer and Account tables, tied together by the Customers' account numbers, as indicated by the foreign entity and key entries for TAccountNo.

Below, is the PTG template for the SQL code that identifies records that exist in one instance (DL1—the “Before” image) and not the other (DL2—the “After” image,) or vice-versa and the record pairs in which one or more data values are different:

```

##
${EntityName} in ('Italy_Name_Addr','AcctTransactions')$
[
@EXTEND
Derive PKColumnName isa ColumnName where PK='PK';
Derive NonStringColumnName isa ColumnName where DataType not in
('varchar,nvarchar');
@
[
-- Check delta for ${EntityName}$
[
select '${EntityName}$' as EntityName,'DELETED ROWS ' ,[a.${ColumnName}$^,\n#03]
from DL1..${EntityName}$ a join

[ (select [${PKColumnName}$^,]
from DL1..${EntityName}$ a
except
select [${PKColumnName}$^,]
from DL2..${EntityName}$ b ] )x
on [a.${PKColumnName}$=x.${PKColumnName}$ ^ and ]

]
UNION ALL
[
select '${EntityName}$' as EntityName,'CHANGED ROWS' ,
[ case a.${ColumnName}$ when b.${ColumnName}$ then '' else '****' end +
[convert(varchar(100),${NonStringColumnName}$)a.${ColumnName}$[${NonStringColumnName}$]]
as ${ColumnName}$ ^,\n#3]
from DL1..${EntityName}$ a join DL2..${EntityName}$ b
on [a.${PKColumnName}$=b.${PKColumnName}$ ^ and ]
where [(a.${ColumnName}$<>b.${ColumnName}$ or
a.${ColumnName}$ is null and b.${ColumnName}$ is not null or
b.${ColumnName}$ is null and a.${ColumnName}$ is not null ) ^ or \n]
union
select '${EntityName}$' as EntityName,'CHANGED ROWS' ,
[ case b.${ColumnName}$ when a.${ColumnName}$ then '' else '****' end +
[convert(varchar(100),${NonStringColumnName}$)b.${ColumnName}$[${NonStringColumnName}$]]
as ${ColumnName}$ ^,\n#3]
from DL1..${EntityName}$ a join DL2..${EntityName}$ b
on [a.${PKColumnName}$=b.${PKColumnName}$ ^ and ]
where [(a.${ColumnName}$<>b.${ColumnName}$ or
a.${ColumnName}$ is null and b.${ColumnName}$ is not null or
b.${ColumnName}$ is null and a.${ColumnName}$ is not null ) ^ or \n]

]
UNION ALL
[
select '${EntityName}$' as EntityName,'INSERTED ROWS ' ,[a.${ColumnName}$^,\n#03]
from DL2..${EntityName}$ a join

[ (select [${PKColumnName}$^,]
from DL2..${EntityName}$ a
except
select [${PKColumnName}$^,]
from DL1..${EntityName}$ b ] )x
on [a.${PKColumnName}$=x.${PKColumnName}$ ^ and ]

]]
]

```

And here are some sample sections of the generated code. First, the code that detects missing rows:

```
-- Check delta for Italy_Name_Addr

select 'Italy_Name_Addr' as EntityName, 'DELETED ROWS '
,a.AccountNo,a.BirthCity,a.Birthday,

a.CF,a.City,a.EmailAddress,

a.Gender,a.GivenName,a.MiddleInitial,

a.PostalCode,a.Provincia,a.Rownum,

a.StreetAddress,a.Surname,a.TelephoneNumber
from DL1..Italy_Name_Addr a join

(select AccountNo
from DL1..Italy_Name_Addr a
except
select AccountNo
from DL2..Italy_Name_Addr b )x
on a.AccountNo=x.AccountNo
```

Then, a sample of the code that identifies and highlights columns with different values:

```
UNION ALL

select 'Italy_Name_Addr' as EntityName, 'CHANGED ROWS' ,
case a.AccountNo when b.AccountNo then '' else '****' end +
convert(varchar(100),a.AccountNo) as AccountNo , case a.BirthCity when b.BirthCity
then '' else '****' end + convert(varchar(100),a.BirthCity) as BirthCity , case
a.Birthday when b.Birthday then '' else '****' end + convert(varchar(100),a.Birthday)
as Birthday ,

case a.CF when b.CF then '' else '****' end +
convert(varchar(100),a.CF) as CF , case a.City when b.City then '' else '****' end +
convert(varchar(100),a.City) as City , case a.EmailAddress when b.EmailAddress then
'' else '****' end + convert(varchar(100),a.EmailAddress) as EmailAddress ,
case a.Gender when b.Gender then '' else '****' end +
convert(varchar(100),a.Gender) as Gender , case a.GivenName when b.GivenName then ''
else '****' end + convert(varchar(100),a.GivenName) as GivenName , case
a.MiddleInitial when b.MiddleInitial then '' else '****' end +
convert(varchar(100),a.MiddleInitial) as MiddleInitial ,
case a.PostalCode when b.PostalCode then '' else '****' end +
convert(varchar(100),a.PostalCode) as PostalCode , case a.Provincia when b.Provincia
then '' else '****' end + convert(varchar(100),a.Provincia) as Provincia , case
a.Rownum when b.Rownum then '' else '****' end + convert(varchar(100),a.Rownum) as
Rownum ,

case a.StreetAddress when b.StreetAddress then '' else '****' end
+ convert(varchar(100),a.StreetAddress) as StreetAddress , case a.Surname when
b.Surname then '' else '****' end + convert(varchar(100),a.Surname) as Surname , case
a.TelephoneNumber when b.TelephoneNumber then '' else '****' end +
convert(varchar(100),a.TelephoneNumber) as TelephoneNumber
from DL1..Italy_Name_Addr a join DL2..Italy_Name_Addr b
on a.AccountNo=b.AccountNo
where (a.AccountNo<>b.AccountNo or
a.AccountNo is null and b.AccountNo is not null or
b.AccountNo is null and a.AccountNo is not null ) or
(a.BirthCity<>b.BirthCity or
a.BirthCity is null and b.BirthCity is not null or
b.BirthCity is null and a.BirthCity is not null ) or
(a.Birthday<>b.Birthday or
a.Birthday is null and b.Birthday is not null or
b.Birthday is null and a.Birthday is not null ) or
(a.CF<>b.CF or
```

And finally, code that identifies inserted rows:

```

UNION ALL

    select 'Italy_Name_Addr' as EntityName, 'INSERTED ROWS '
    , a.AccountNo, a.BirthCity, a.Birthday,

    a.CF, a.City, a.EmailAddress,

    a.Gender, a.GivenName, a.MiddleInitial,

    a.PostalCode, a.Provincia, a.Rownum,

    a.StreetAddress, a.Surname, a.TelephoneNumber
    from DL2..Italy_Name_Addr a join

(select AccountNo
 from DL2..Italy_Name_Addr a
 except
 select AccountNo
 from DL1..Italy_Name_Addr b )x
 on a.AccountNo=x.AccountNo

```

Here is the metadata for the second database example, housed in the same specification structure as the previous example:

1	EntityName	ColumnName	DataType	Length	PK	ForeignEntityName	FPK
82	V_DA_SALES_POSITION	REGION_OFFICE_ADDRESS1	nvarchar	510			
83	V_DA_SALES_POSITION	REGION_OFFICE_ADDRESS2	nvarchar	510			
84	V_DA_SALES_POSITION	REGION_OFFICE_ADDRESS3	nvarchar	510			
85	V_DA_SALES_POSITION	REGION_OFFICE_CITY	nvarchar	510			
86	V_DA_SALES_POSITION	REGION_OFFICE_ID	nvarchar	510			
87	V_DA_SALES_POSITION	REGION_OFFICE_PHONE_NUM	nvarchar	510			
88	V_DA_SALES_POSITION	REGION_OFFICE_STATE	nvarchar	510			
89	V_DA_SALES_POSITION	REGION_OFFICE_ZIPCODE	nvarchar	510			
90	V_DA_SALES_POSITION	ROLE_CD	nvarchar	510			
91	V_DA_SALES_POSITION	ROLE_DESC	nvarchar	510			
92	V_DA_SALES_POSITION	ROLE_ID	nvarchar	510			
93	V_DA_SALES_POSITION	SALES_ORGANIZATION_CD	nvarchar	510			
94	V_DA_SALES_POSITION	SALES_ORGANIZATION_DESC	nvarchar	510			
95	V_DA_SALES_POSITION	SALES_POSITION_DESC	nvarchar	510			
96	V_DA_SALES_POSITION	SALES_POSITION_ID	nvarchar	510	PK		
97	V_DA_SALES_POSITION	SALES_POSITION_TYPE_CD	nvarchar	510			
98	V_DA_SALES_POSITION	SALES_POSITION_TYPE_DESC	nvarchar	510			
99	V_DA_SALES_POSITION	SAMPLES_FG	nvarchar	510			
100	V_DA_SALES_POSITION	SFA_DB	nvarchar	510			
101	V_DA_SALES_POSITION	VACANT_FG	nvarchar	510			
102	V_DA_SALES_POSITION_MATRIX	RELATED_SALES_POSITION_ID	nvarchar	510			
103	V_DA_SALES_POSITION_MATRIX	RELATIONSHIP_TYPE_CD	nvarchar	510	PK		
104	V_DA_SALES_POSITION_MATRIX	RELATIONSHIP_TYPE_DESC	nvarchar	510			
105	V_DA_SALES_POSITION_MATRIX	SALES_POSITION_ID	nvarchar	510	PK		
106	V_DA_SALES_POSITION_PRODUCT	PRODUCT_BRAND_ID	nvarchar	510	PK		
107	V_DA_SALES_POSITION_PRODUCT	PRODUCT_DESC	nvarchar	510			
108	V_DA_SALES_POSITION_PRODUCT	SALES_POSITION_ID	nvarchar	510	PK		
109	Italy_Name_Addr	AccountNo	int				

And, here is the part of the code that detects deleted rows that was generated from it:

```

-- Check delta for V_DA_PERSON

select 'V_DA_PERSON' as EntityName, 'DELETED ROWS '
, a.ACCEPT_SHIPMENTS_FG, a.AIRPORT, a.BULK_ADDRESS1,
a.BULK_ADDRESS2, a.BULK_CITY, a.BULK_PHONE,
a.BULK_POSTAL_CD, a.BULK_STATE, a.BULK_ZIP,
a.BULK_ZIP4, a.CARCO_DT, a.CARCO_FG,
a.CELL_PHONE, a.DIST_COMP_EXPERT_FLAG, a.EFFECTIVE_HIRE_DATE,
a.EMAIL_ADDRESS, a.EMPL_SOURCE_CD, a.EMPL_STATUS,
a.EMPL_STATUS_DESC, a.EMPLID, a.EQUIPMENT_DT,
a.ETHNICITY, a.FAX_PHONE, a.FIELD_ACTIVE,
a.FIELD_ACTIVE_DESC, a.FIRST_NAME, a.FULL_NAME,
a.GENDER, a.HIRE_DATE, a.HOME_ADDRESS1,
a.HOME_ADDRESS2, a.HOME_AREA_CODE, a.HOME_CITY,
a.HOME_PHONE, a.HOME_POSTAL_CD, a.HOME_STATE,
a.HOME_ZIP, a.HOME_ZIP4, a.IMS_DOMAIN,
a.IMS_FUID, a.IMS_GUID, a.JOB_CODE,
a.JOB_DESCRIPTION, a.JOB_TITLE, a.LAST_NAME,
a.MAIL_ADDRESS1, a.MAIL_ADDRESS2, a.MAIL_CITY,
a.MAIL_STATE, a.MAIL_ZIP, a.MARITAL_STATUS,
a.MASTERS_FG, a.MIDDLE_NAME, a.MILITARY_BRANCH,
a.MILITARY_END_DATE, a.MILITARY_RANK, a.MILITARY_START_DATE,
a.PENDING_TRANSACTION_TYPE, a.PREFERRED_NAME, a.PREFIX,
a.REHIRE_DATE, a.SALES_POSITION_ID, a.SOURCE,
a.TERMINATION_DATE, a.VAC_CHG, a.VOICE_MAIL,
a.WORK_STATUS

from DL1..V_DA_PERSON a join
(select EMPLID
from DL1..V_DA_PERSON a
except
select EMPLID
from DL2..V_DA_PERSON b )x
on a.EMPLID=x.EMPLID

```

Here is the output that results from running the generated code. Each pair of rows shows the analogous rows from both sources, as determined by matching their primary keys, and differences in column values are highlighted by four asterisks:

db_source	EntityName	Status	ACCEPT_SHIPMENTS_FG	AIRPORT	BULK_ADDRESS1	BULK_ADDRESS2	BULK_CITY	BULK_PHONE	BULK_ZIP
1	DL1	V_DA_PERSON	CHANGED ROWS	Y	NULL	****Changed	16714 E. Sprague Avenue	Spokane Valley	5094872772
2	DL2	V_DA_PERSON	CHANGED ROWS	Y	NULL	****Veradale Self Storage	16714 E. Sprague Avenue	Spokane Valley	5094872772
3	DL1	V_DA_PERSON	CHANGED ROWS	N	NULL	Storage USA-Unit 3070	201 64th Street	****Brooklyn South	7187484499
4	DL2	V_DA_PERSON	CHANGED ROWS	N	NULL	Storage USA-Unit 3070	201 64th Street	****Brooklyn	7187484499
5	DL1	V_DA_PERSON	CHANGED ROWS	Y	NULL	Storage USA	201 64th Street Unit 3149	****Brooklyn South	7187484499
6	DL2	V_DA_PERSON	CHANGED ROWS	Y	NULL	Storage USA	201 64th Street Unit 3149	****Brooklyn	7187484499
7	DL1	V_DA_PERSON	CHANGED ROWS	Y	NULL	Soma Self Storage	1475 Mission St.	****Brooklyn South	7187484499
8	DL2	V_DA_PERSON	CHANGED ROWS	Y	NULL	Soma Self Storage	1475 Mission St.	****San Francisco	7187484499
9	DL1	V_DA_PERSON	CHANGED ROWS	Y	NULL	Extra Space Storage Unit	201 64th St. Unit 3110	****Brooklyn South	7187484499
10	DL2	V_DA_PERSON	CHANGED ROWS	Y	NULL	Extra Space Storage Unit	201 64th St. Unit 3110	****Brooklyn	7187484499
11	DL1	V_DA_PERSON	CHANGED ROWS	Y	NULL	201 64th Street	Unit #3006	****Brooklyn South	7187484499
12	DL2	V_DA_PERSON	CHANGED ROWS	Y	NULL	201 64th Street	Unit #3006	****Brooklyn	7187484499

Once the specifications (extracted from the database system tables) and the template were in place for the initial solution, the second instance for the five-table example was generated from the GES in literally, a moment.

## Dojo Browser-based Database Interface

**Use Case:** Create a browser-based interface that provides data browsing and editing of a relational database with support for parent-child entities. This solution would be of value to anyone that needs to maintain RDBMS data for which a pre-built GUI interface does not exist.

**Functionality:** The solution allows a user to filter, browse, insert, update or delete data in the database. The data presentation automatically adjusts to account for table-level relationships represented as constraints in the database or defined in enrichment data in the Specifications.

Solution features include:

- a menu for accessing main screens,
- tabular or single screens for each table for viewing, adding, deleting or updating table rows,
- data selection screens and drop down menu for type-bound attributes,
- validation checks at the record level,
- search and filter services,
- parent- child navigation links,
- parent-child and selector-entity composite displays, which appear as a parent record with child rows in linked tabular views or a selected object and its entity details in a side panel and
- history or log table generation with supporting code to manage database transactions.

**Metadata:** The metadata for this case consists of the metadata extracted from the database system tables. Enrichment data consists of information used to identify composite entities (logical entities represented as multiple columns or rows in a number of related tables) and table- or entity-level business rules.

**Technology:** This solution was implemented as an MVC architecture in Java web technology with a Dojo interface layer.

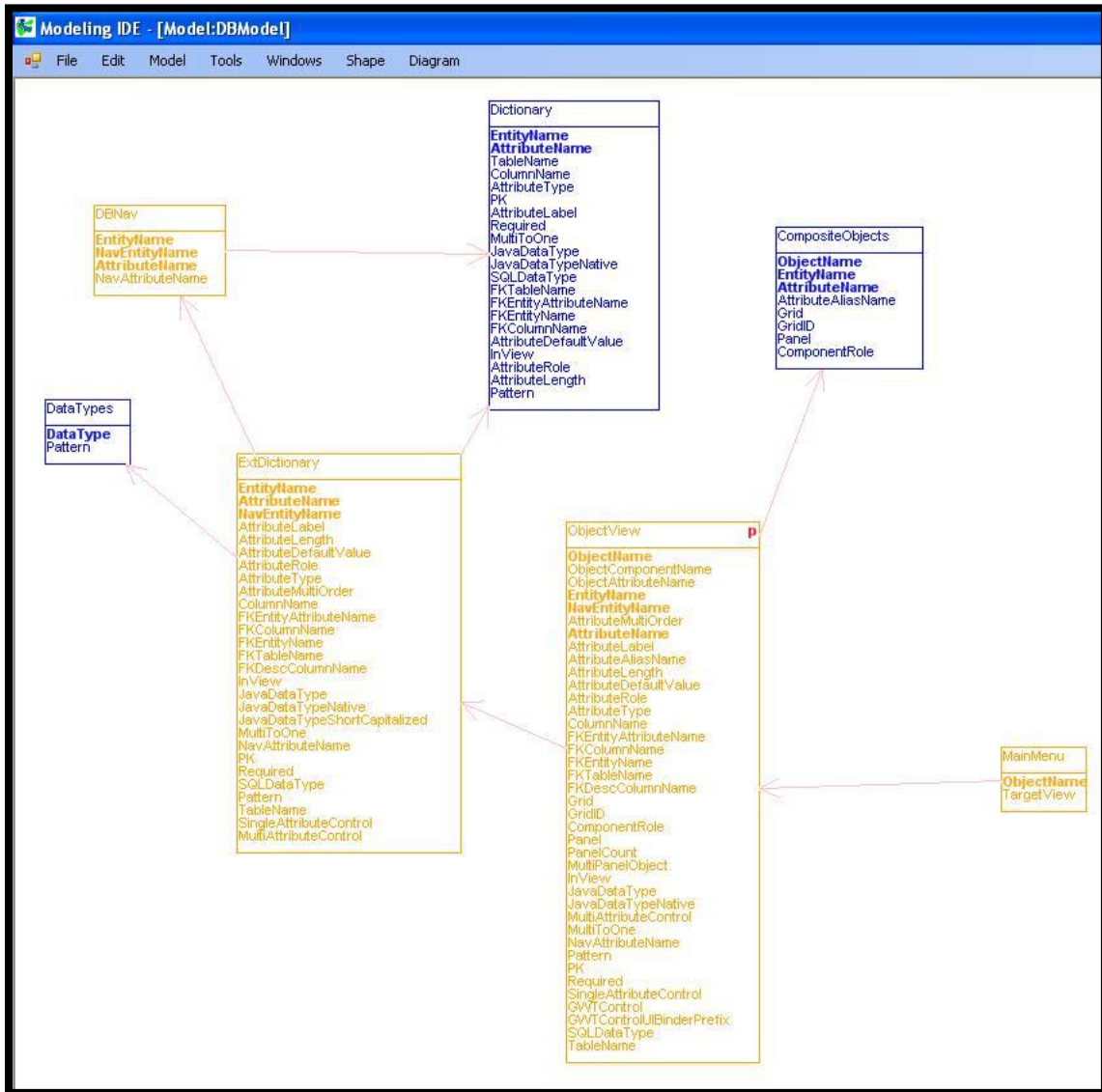
This solution is a good example of how the interplay between enhanced specification data and templates can be used to maintain architectural focus and minimize development time and maintenance effort over the life of the solution.

The base specification data consists of the data structure of the database. The enhancement data consists of identifying navigation links, identifying composite entities and assigning them to a specific visual display objects in the application interface.

The templates create code that supports data selection, navigation, and display.



Here is a diagram of the specification data model. Note that the tables in blue denote database metadata and the tables in orange denote enhancement data:



The screenshot, below shows the database metadata and some of the enhancement data behind the specification data model:

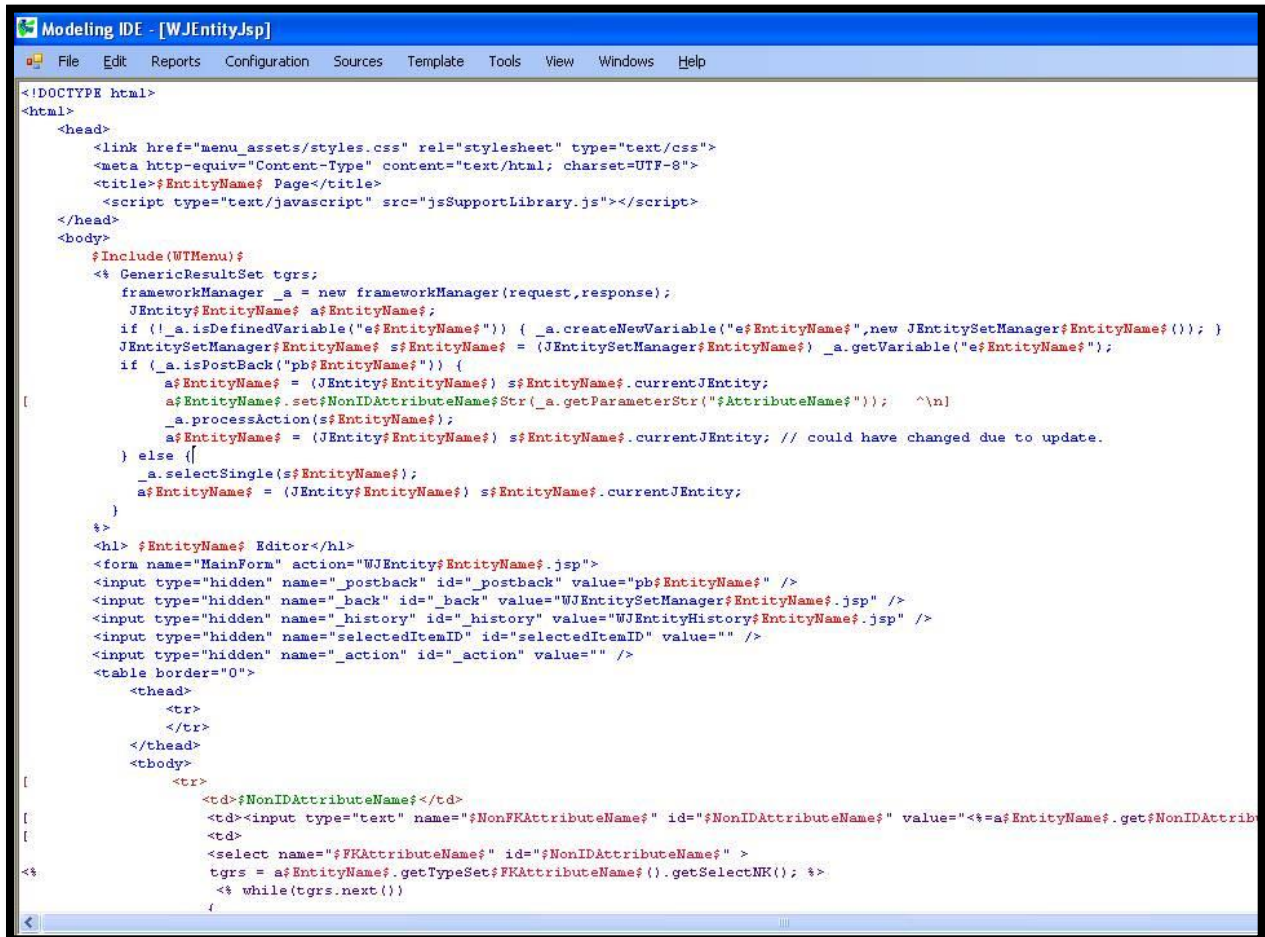
ObjectName	AttributeName	AttributeLabel	GWTControl
CustomerPanel	Addressline1	Addressline1	TextBox
CustomerPanel	Addressline2	Addressline2	TextBox
CustomerPanel	City	City	TextBox
CustomerPanel	CreditLimit	CreditLimit	TextBox
CustomerOrder	CustomerId	Customer	TextBox
CustomerView	CustomerId	Customer	TextBox
CustomerPanel	CustomerId	Customer	TextBox
CustomerPanel	DiscountCode	DiscountCode	CListBox
CustomerPanel	Email	Email	TextBox
CustomerPanel	Fax	Fax	TextBox
CustomerOrder	lName	Name	TextBox
CustomerView	lName	Name	TextBox
CustomerPanel	lName	Name	TextBox
CustomerPanel	Phone	Phone	TextBox
CustomerPanel	State	State	TextBox
CustomerPanel	Zip	Zip	TextBox
CustomerView	Fax	Fax	TextBox
CustomerView	Phone	Phone	TextBox
CustomerView	State	State	TextBox
CustomerView	CreditLimit	CreditLimit	TextBox
CustomerView	Addressline1	Addressline1	TextBox
CustomerView	Addressline2	Addressline2	TextBox
CustomerView	City	City	TextBox
CustomerView	Email	Email	TextBox
CustomerView	DiscountCode	DiscountCode	CListBox
CustomerView	Zip	Zip	TextBox
CustomerOrder	SalesDate	SalesDate	TextBox
CustomerOrder	ShippingDate	ShippingDate	TextBox

PK	Group_4	CONST	ObjectName	ObjectComponentName	ObjectAttributeName	EntityName
NavEntityName	AttributeMultiOrder	AttributeName	AttributeLabel	AttributeAliasName	AttributeLength	AttributeDefaultValue
AttributeRole	AttributeType	ColumnName	FKEntityAttributeName	FKColumnName	FKEntityName	FKTableName
FKDescColumnName	Grid	GridID	ComponentRole	Panel	PanelCount	MultiPanelObject
InView	JavaDataType	JavaDataTypeNative	MultiAttributeControl	MultiToOne	NavAttributeName	Pattern
PK	Required	SingleAttributeControl	GWTControl	GWTControlUIBinder	SQLDataType	TableName



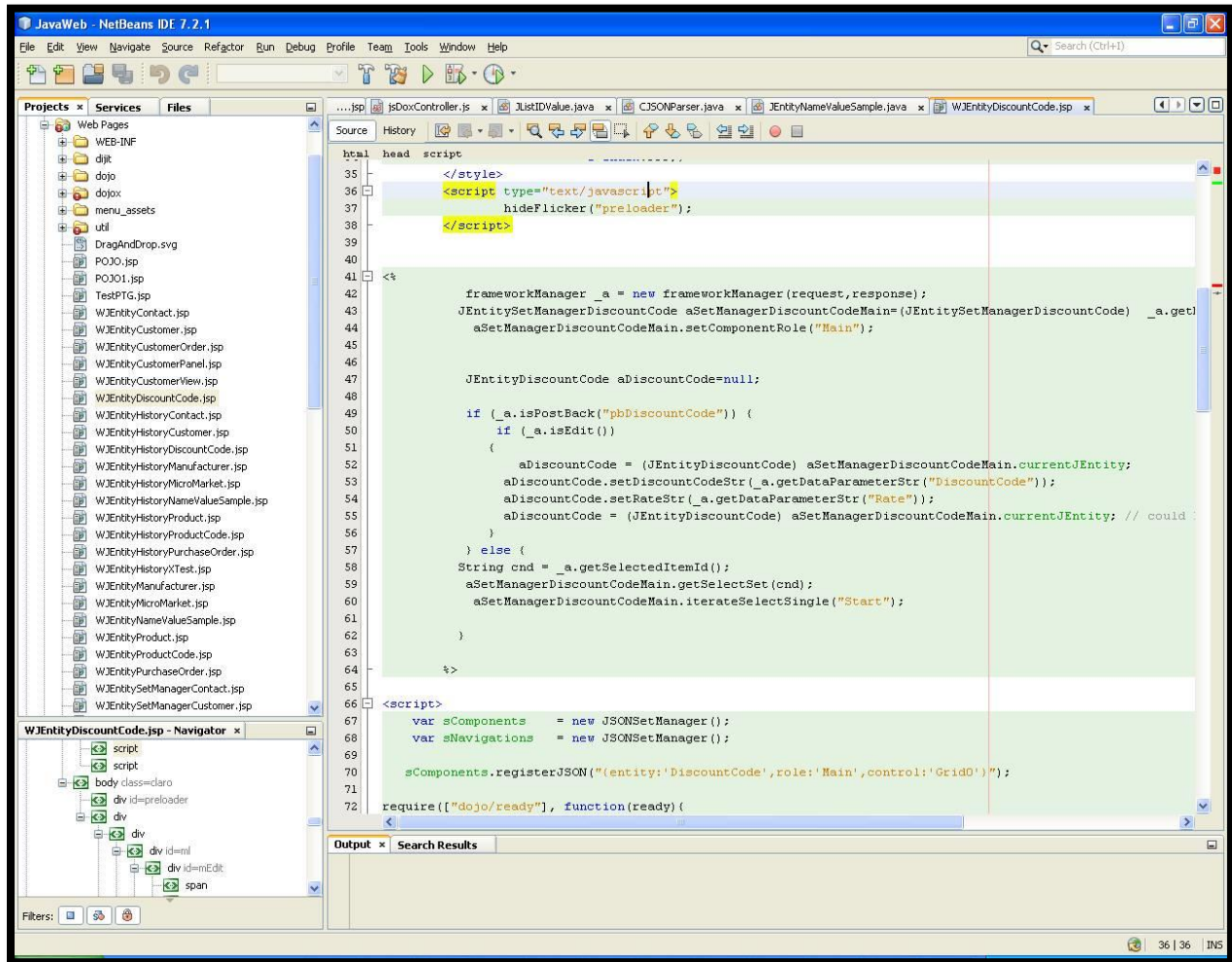
Here is one of the templates from which Dojo code is generated:



```
Modeling IDE - [WJEntityJsp]
File Edit Reports Configuration Sources Template Tools View Windows Help

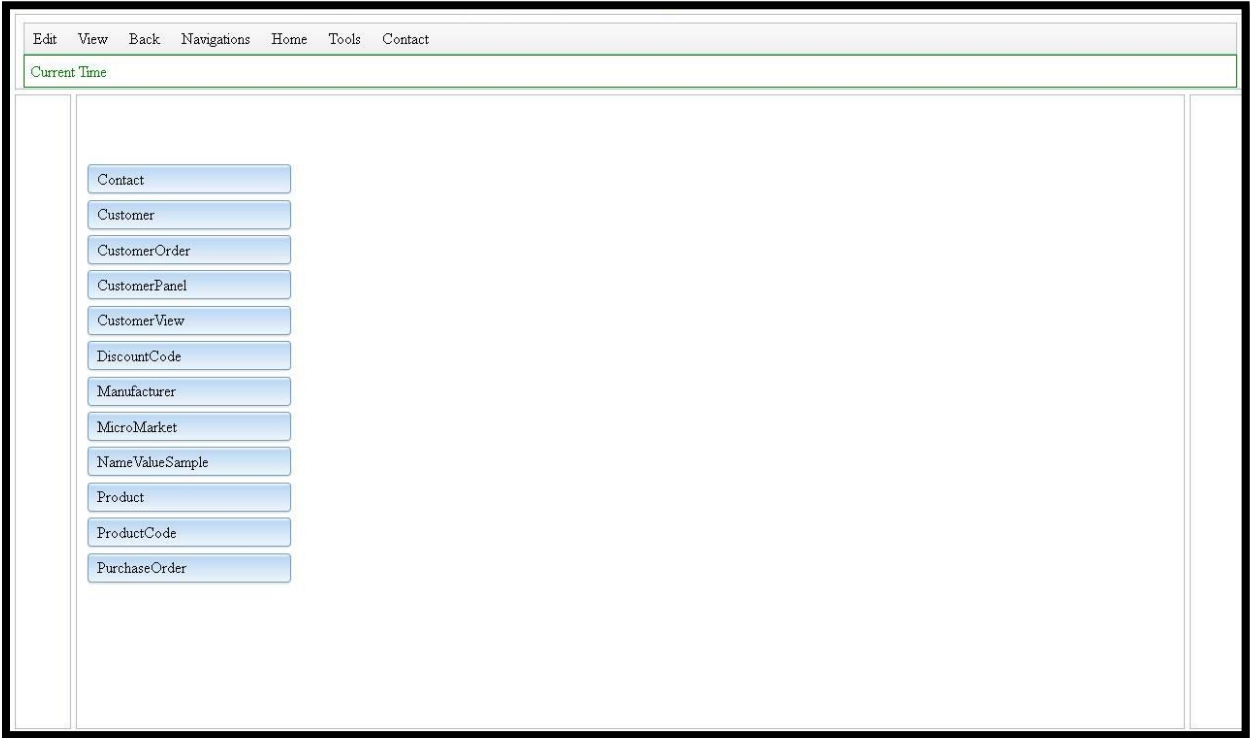
<!DOCTYPE html>
<html>
  <head>
    <link href="menu_assets/styles.css" rel="stylesheet" type="text/css">
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <title>${EntityName} Page</title>
    <script type="text/javascript" src="jsSupportLibrary.js"></script>
  </head>
  <body>
    <include(WTMenu)>
    <!-- GenericResultset tgrs;
    frameworkManager _a = new frameworkManager(request,response);
    JEntity ${EntityName} a=${EntityName};
    if (!_a.isDefinedVariable("e${EntityName}") ) { _a.createNewVariable("e${EntityName}",new JEntitySetManager ${EntityName}()); }
    JEntitySetManager ${EntityName} s=${EntityName} = (JEntitySetManager ${EntityName}) _a.getVariable("e${EntityName}");
    if (_a.isPostBack("pb${EntityName}")) {
      a=${EntityName} = (JEntity ${EntityName}) s=${EntityName}.currentJEntity;
      a=${EntityName}.setNonIDAttributeNameStr(_a.getParameterStr("${AttributeName}")); ^\n
      _a.processAction(s=${EntityName});
      a=${EntityName} = (JEntity ${EntityName}) s=${EntityName}.currentJEntity; // could have changed due to update.
    } else {
      _a.selectSingle(s=${EntityName});
      a=${EntityName} = (JEntity ${EntityName}) s=${EntityName}.currentJEntity;
    }
    -->
    <h1> ${EntityName} Editor</h1>
    <form name="MainForm" action="WJEntity ${EntityName}.jsp">
    <input type="hidden" name="_postback" id="_postback" value="pb${EntityName}" />
    <input type="hidden" name="_back" id="_back" value="WJEntitySetManager ${EntityName}.jsp" />
    <input type="hidden" name="_history" id="_history" value="WJEntityHistory ${EntityName}.jsp" />
    <input type="hidden" name="selectedItemID" id="selectedItemID" value="" />
    <input type="hidden" name="_action" id="_action" value="" />
    <table border="0">
      <thead>
        <tr>
          <td></td>
        </tr>
      </thead>
      <tbody>
        <tr>
          <td><NonIDAttributeName></td>
          <td><input type="text" name="${NonFKAttributeName}" id="${NonIDAttributeName}" value="<%=a=${EntityName}.getNonIDAttrib
          <td>
          <select name="${FKAttributeName}" id="${NonIDAttributeName}" >
            tgrs = a=${EntityName}.getTypeSet ${FKAttributeName}().getSelectNK(); %>
            <%= while(tgrs.next())
              <
            </%>
          </td>
        </tr>
      </tbody>
    </table>
  </body>
</html>
```

Here is a screenshot from the JavaBean IDE showing the scope of the Dojo code generated by the GES:



Finally, below are a series of screenshots of the generated solution:

Main Menu:



### Customer, single record view:

Edit View Back Navigations Home Tools Contact

Current Time

Name	Small Bill Company
Addressline1	8585 South Upper Murray Drive
Addressline2	P.O. Box 456
City	Alanta
CreditLimit	80000
DiscountCode	L
Email	www.smallbill.example.com
Fax	555-555-0176
Phone	555-555-0175
State	GA
Zip	12347

### Customer, multi-record view:

Edit View Back Navigations Home Tools Contact

Current Time

Name	Addressline1	Addressline2	City	CreditLimit	DiscountCode	
Jumbo Eagle Corp	28 EverGreen Place 2	Suite 51240000	Fort Lauderdale	20000	M	jumboeagle@ex
New Enterprises	9754 Main Street	P.O. Box 5678	Miami	10000	N	www.new.exa
Small Bill Company	8585 South Upper Murray Drive	P.O. Box 456	Alanta	80000	L	www.smallbill.e
Wren Computers	8989 Red Albatross Drive	Suite 9897	Houston	25000	L	www.wrencor
Bob Hosting Corp.	65653 Lake Road	Suite 2323	San Mateo	65000	H	www.bobhos

## Customer-Order:

Edit View Back Navigations Home Tools Contact

Current Time

**Name**

- Jumbo Eagle Corp
- New Enterprises
- Small Bill Company
- Wren Computers
- Bob Hosting Corp
- Early CentralComp
- John Valley Computers
- Old Media Drink online

Customer	FreightCompany	OrderNum	Product	Quantity	SalesDate	ShippingCost	ShippingDate
Bob Hosting Corp.	Slow Snail	10398006	Printer Cable	60	2011-05-24	55.00	2011-05-24
Bob Hosting Corp.	Slow Snail	10398007	24 inch Digital Monitor	120	2011-05-24	65.00	2011-05-24

## Enhancement of an Interface to Incorporate Temporal Features

**Use Case:** The database used in the previous case has structures (parallel tables in which historical images of transacted records for selected base tables are maintained) that allow for representation of data as it appeared at any time in the past, as it appears currently or as it will appear at any point in the future. The ability to view temporally-filtered data is valuable to users wishing to visualize the state of the data juxtaposed with other events, such as a time series of financial market transactions. The ability to view future states is an enhancement that may be useful for planning and modeling policies to be implemented in the future, such as a proposed realignment of sales territories.

**Functionality:** The requirement for this use case is to add functionality to the previously-built interface that preserves the users' ability to filter, browse, enter or edit data and adds the ability to limit operations to a subset of data as it either did or would appear at a selected point in time.

This solution accommodates significant complexities in terms of the SQL required to support the temporal features. The solution features include:

- all of the interface features of the previous use case,
- data filtering functions that support normal SQL selection semantics and incorporate the ability to apply them to a time-specific subset of the data
- complex joins, which return the relevant records for the join from each table, which adds a significant set of constraints,
- data management filters that ensure that only relevant values appear in drop down and selection controls and
- data transaction operations that update the temporal history for transactions on tables that are logged.

**Metadata:** The metadata and enrichment data for this case are the same as for the previous example. The specifications consist of metadata extracted from the database system tables and enrichment data consists of information used to identify composite entities (logical entities represented in a number of related tables) and table- or entity-level business rules.

**Technology:** This solution was implemented as an MVC architecture in Java web technology with a JavaScript and DOJO interface layer.

The GES interface showing the specification metadata behind the temporal database:

EntityName	AttributeName	TableName	ColumnName	AttributeType	PK	AttributeLabel	Required	MultiToOne	JavaDataType	JavaDataTypeNative	SQLDataType	FKTable
Contact	ContactId	Contact	CONTACT_ID	ID	Y	Id	Y		Integer	Integer	INTEGER	INTEGER
Contact	Firstname	Contact	FIRST_NAME	NK	N	FirstName	Y		String	String	VARCHAR	VARCHA
Contact	Lastname	Contact	LAST_NAME	DATA	N	LastName	Y		String	String	VARCHAR	VARCHA
Contact	EmailAddress	Contact	EMAIL_ADDRESS	DATA	N	EmailAddress	Y		String	String	VARCHAR	VARCHA
Customer	CustomerId	Customer	CUSTOMER_ID	ID	Y	Customer	Y		Integer	Integer	INTEGER	
Customer	DiscountCode	Customer	DISCOUNT_CODE	FK	N	DiscountCode	Y		String	String	CHAR	DISCOUN
Customer	Zip	Customer	ZIP	DATA	N	Zip	Y		String	String	VARCHAR	MICRO_M
Customer	Name	Customer	NAME	NK	N	Name	N		String	String	VARCHAR	
Customer	Addressline1	Customer	ADDRESSLINE1	DATA	N	Addressline1	N		String	String	VARCHAR	
Customer	Addressline2	Customer	ADDRESSLINE2	DATA	N	Addressline2	N		String	String	VARCHAR	
Customer	City	Customer	CITY	DATA	N	City	N		String	String	VARCHAR	
Customer	State	Customer	STATE	DATA	N	State	N		String	String	CHAR	
Customer	Phone	Customer	PHONE	DATA	N	Phone	N		String	String	CHAR	
Customer	Fax	Customer	FAX	DATA	N	Fax	N		String	String	CHAR	
Customer	Email	Customer	EMAIL	DATA	N	Email	N		String	String	VARCHAR	
Customer	CreditLimit	Customer	CREDIT_LIMIT	DATA	N	CreditLimit	N		Integer	Integer	INTEGER	
DiscountCode	DiscountCode	Discount_Code	DISCOUNT_CODE	NK	Y	DiscountCode	Y		String	String	CHAR	
DiscountCode	Rate	Discount_Code	RATE	DATA	N	Rate	N		java.math.BigDecimal	java.math.BigDecimal	DECIMAL	
Manufacturer	ManufacturerId	Manufacturer	MANUFACTURER_ID	ID	Y	Manufacturer	Y		Integer	Integer	INTEGER	
Manufacturer	Name	Manufacturer	NAME	NK	N	Name	N		String	String	VARCHAR	
Manufacturer	Addressline1	Manufacturer	ADDRESSLINE1	DATA	N	Addressline1	N		String	String	VARCHAR	
Manufacturer	Addressline2	Manufacturer	ADDRESSLINE2	DATA	N	Addressline2	N		String	String	VARCHAR	
Manufacturer	City	Manufacturer	CITY	DATA	N	City	N		String	String	VARCHAR	
Manufacturer	State	Manufacturer	STATE	DATA	N	State	N		String	String	CHAR	
Manufacturer	Zip	Manufacturer	ZIP	DATA	N	Zip	N		String	String	CHAR	
Manufacturer	Phone	Manufacturer	PHONE	DATA	N	Phone	N		String	String	VARCHAR	
Manufacturer	Fax	Manufacturer	FAX	DATA	N	Fax	N		String	String	VARCHAR	

Column Selection:  CheckAll  UnCheckAll

Conditions: [Empty]

Number of records = 56

Here is a template that performs temporal data selection:

```

select Id,
  [${~ColumnName$ $AliasAttributeName$ ^, \n#03}
  [,   $AliasAttributeName$_TR ${~FKColumnName$ ^\n]
from (
  select [main.$PKColumnName$ as ID],
  [ $EntityAliasName$. $ColumnName$ $AliasAttributeName$ ^\n, #02]
  [, $FKEntityName$. $FKDescColumnName$ $AliasAttributeName$_TR ${~FKColumnName$ ^\n]
,main.ts,main.trn
from $TableName$_HIST main
[
  left join (select * from $FKTableName$_Hist
              where ts = (select max(ts) from $FKTableName$_Hist h$FKTableName$
                          where [$FKTableName$_Hist.$FKColumnName$=h$FKTableName$. $FKColumnName$] and ts<=@TS@ )
              and trn='I'
              ) $FKEntityName$ on Main.$ColumnName$=$FKEntityName$. $FKColumnName$ ^\n]
where main.ts = (select max(ts) from $TableName$_hist h
                 [where main.$PKColumnName$=h.$PKColumnName$ and ts<=@TS@] )
and main.trn='I'
) x
@CONDITIONS@
[
  order by $NKAttributeName$ ^,]

```

And here is some of the SQL code generated from it:

```
select OID,Id,ts,trn,
    Available, Description, ManufacturerId,
    Markup, ProductCode, ProductId,
    PurchaseCost, QuantityOnHand
,      ManufacturerId_TR
,      ProductCode_TR
from (
select main.PRODUCT_ID as OID,main.ID as ID,
    main.AVAILABLE Available , main.DESCRPTION Description
main.MANUFACTURER_ID ManufacturerId , main.MARKUP Markup
main.PRODUCT_CODE ProductCode , main.PRODUCT_ID ProductId
main.PURCHASE_COST PurchaseCost , main.QUANTITY_ON_HAND QuantityOnHand
, Manufacturer.NAME ManufacturerId_TR
ProductCode.DESCRPTION ProductCode_TR
,main.ts,main.trn
from Product HIST main
    left join (select * from MANUFACTURER_Hist
        where ts = (select max(ts) from MANUFACTURER_Hist hMANUFACTURER
            where MANUFACTURER_Hist.MANUFACTURER_ID=hMANUFACTURER.MANUFACTURER_ID and ts<main.ts )
        and trn='I'
        ) Manufacturer on Main.MANUFACTURER_ID=Manufacturer.MANUFACTURER_ID
    left join (select * from PRODUCT_CODE_Hist
        where ts = (select max(ts) from PRODUCT_CODE_Hist hPRODUCT_CODE
            where PRODUCT_CODE_Hist.PROD_CODE=hPRODUCT_CODE.PROD_CODE and ts<main.ts )
        and trn='I'
        ) ProductCode on Main.PRODUCT_CODE=ProductCode.PROD_CODE
) x
order by Id
```

Here is an example of temporally-filtered output from the enhanced solution, a Customer record, with timestamp selection:

The screenshot shows a web application interface with a menu bar (Edit, View, Back, Navigations, Home, Tools, Contact) and a timestamp 'As of 15/03/2013'. Below the menu, there are two tabs: 'Address' (selected) and 'Information'. The 'Address' tab displays a form with the following fields and values:

Addressline1	28 EverGreen Place 2
Addressline2	Suite 51240000
City	Fort Lauderdale
Email	jumboeagle@example.com
Fax	305-555-0189
Phone	305-555-0188
State	KK
Zip	95117



## Browser-Editor for COBOL File Data

**Use Case:** There is a tremendous amount of data residing in COBOL-formatted files. While mainframe browsing and editing tools exist, many organizations would be well served to have a solution accessible through a standard internet browser.

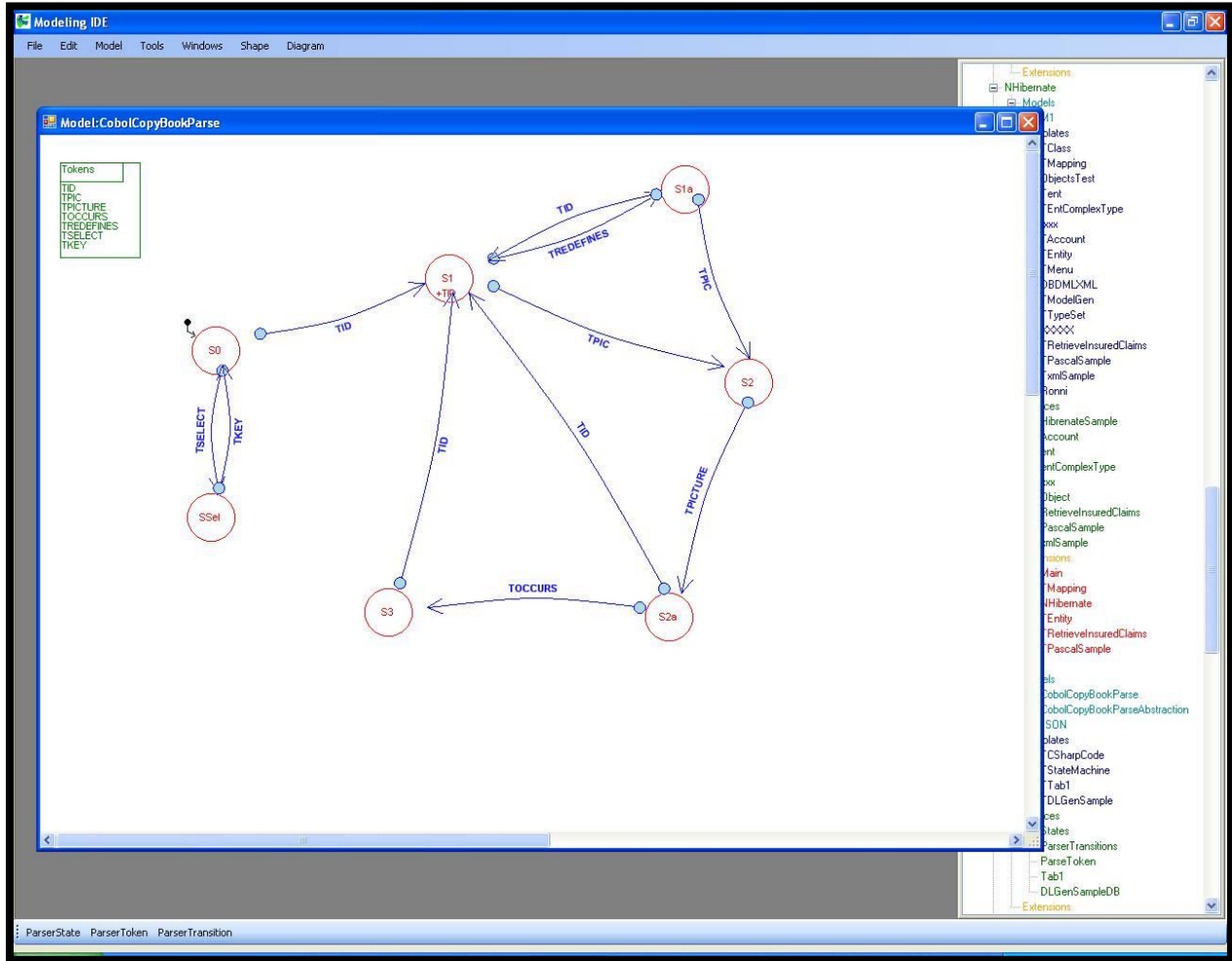
**Functionality:** This solution provides the ability to parse a copybook and populate the specifications with metadata extracted from it in order to generate a program that has the ability to parse, view and edit data stored in native COBOL files in a browser-based GUI interface.

**Metadata:** The metadata for this case consists of relational, tabular or XML data extracted from COBOL copybooks. The parser that extracts the metadata is represented in a state machine diagram that after relational transformation becomes a program that can parse copybooks and present data extracted from COBOL data files.

**Specification Enrichment Data:** None was required in this case.

**Technology:** C# GUI interface.

The diagram below depicts a state transition diagram that is used to control the parsing of the relationships contained in a COBOL copybook:



The diagram below shows the C# interface displaying a record from the COBOL file:

The screenshot shows a window titled "Menu" with a blue border. At the top, there are navigation buttons: "<<", "<", ">", ">>", and "Back". To the right of these are three tabs: "FGEMPL", "FGCOMP", and "FGDEPT". The main area contains a form with the following fields:

FGE-EMP-KEY	FGE-COMPANY	01	
FGE-EMP-KEY	FGE-NUMBER	10034	
FGE-EMP-KEY	FGE-REC-TYPE	M	
FGE-ALT-KEY	FGE-LASTNAME	MAREE	
FGE-INITIALS		AA	
FGE-TITLE		1	
FGE-SEX		M	
FGE-MARRIED		Y	
FGE-SSNUMBER		498-7482741-43	
FGE-ADDR	FGE-STREET	3 MECURIUS ST SUNNYSIDE	FGE-EMPREC.FGE-ADDR
FGE-ADDR	FGE-CITYSTATE		
FGE-ADDR	FGE-ZIP-CODEX	02761 343-6543	FGE-EMPREC.FGE-ADDR.FGE-ZIP(X)
FGE-PHONE		343-6543	
FGE-FROM-DTE		19920301	
FGE-DEPT-NUM		01	
FGE-BITMAP		FGMAN1	
FGE-DOB		19711012	
FGE-DLM		19990301	
FILLER			

## Summary

The use-cases presented in this whitepaper each demonstrate the benefits we identified in the Introduction. In each case, employing CodiScent tools and methodology enhanced the return on the investment in implementing the solution. Here are some highlights of how benefits were realized in some of the use-cases:

### Coding Leverage

- This benefit accrues in every case. In the Database Compare use-case, the template, which generates SQL code to identify records not in both tables or records containing differing values, was 58 lines. 256 lines of code were generated from it for the two-table example and 1,762 lines of code were generated for the five-table example. The generated to hand-written ratio and percentage of generated code for the examples are 4.6:1 and 81.5% and 30.4:1 and 96.8%, respectively.
- In the Dojo Database Interface and Temporal Extension use-cases, a variety of code, including SQL, Java and Java Script was generated at ratios exceeding 98% of the total code base.

### Reuse

- The Database Compare use-case is an excellent example of CodiScent solution reusability. The generation process and its ability to produce rapid, evolutionary, full-scale iterations contributed to the ability to prototype, run, test and revise the solution as it was developed and ultimately produce an error-free implementation. Overall, this solution took less than two hours to develop and less than five minutes to replicate for a second instance.
- Reusability is also represented in the Temporal Extension use-case. When enhancing the solution to incorporate temporal filtering, the coders were able to revise the code layer that communicates with the database and make minimal modifications to the presentation layer.

### Architectural Focus

- The Dojo Database Browser demonstrates this benefit in that the implementation closely maps to the MVC model. It also provides good separation of responsibilities, which would allow programmers to change the presentation of a logical entity quickly and easily by modifying the specification data or structure without revising the templates.

### Rolling Refactoring

- In the Database Compare use-case the initial solution was built to identify rows in which changes had occurred and then enhanced to identify the columns in which they had occurred. Modifying the solution templates and then regenerating the entire solution automatically refactored the result.

### Reduced Maintenance Effort

- The two versions of the Database Compare use-case demonstrate how CodiScent generative engineering can minimize maintenance effort. The initial version of the solution identified records that were missing or had changed values. The second version contained code to annotate the columns in which the differing values occur. The maintenance effort to implement the second version was limited to the SQL templates and was completely independent of any of the other solution components. Similarly, the implementation of the second solution instance,

consisting of the five-table example, was nearly *maintenance-free*, requiring only regeneration of the specification dataset.

### Staffing Leverage

- Although both solution instances in The Database Compare use-case were implemented by the same programmer, it could have easily been someone else. This would have allowed the original developer to focus on solving other problems while the second solution instance was implemented.
- In the Dojo Database Browser use-case, templates used to generate Dojo interface code can be shared and reused by multiple programming groups in a black box fashion. Therefore, the templates can provide significant Staffing Leverage.

### Migration and Transformation

- None of the use-cases are examples of Migration and Transformation, specifically; however, it is easy to see how the Architectural Focus and extensibility of the CodiScent tools lend themselves to supporting them. Such solutions require interfaces to the source and target data repositories and a transformation engine in between and CodiScent generative solutions are perfect for creating them. If the structure of either the source or target change, the solution can be modified quickly to adjust the interface on either side, revise data transformations or revise the source to target map, as necessary.

## Working With CodiScent

Given the proven value of employing CodiScent tools, methodology and services to deliver solutions, what remains to be considered is how to select the best model to meet the need in a particular situation. The service models cited in the initial whitepaper provide a range of options with varying benefits:

- **Turnkey Development** is most appropriate for implementing a solution with the expectation of infrequent updates or modifications. The benefit of Better, Cheaper and Faster development is achieved in terms of the initial cost to implement and opportunities to realize the secondary benefits of reduced maintenance and ability to migrate or transform the solution are preserved as long as the original solution artifacts are maintained.
- **Wizard Implementation** is appropriate for implementing a solution with a moderate expectation of updates or modifications and/or the desire to be able to generate additional solution instances within a limited specification domain. As a result, Reusability and Architectural Focus play a role in the delivered solution. In addition to the benefits associated with Turnkey Development, reduced maintenance, an enhanced ability to migrate or transform the solution and staffing leverage are also achieved.
- **Productization** is a combination of consultive and implementation services. In addition to rationalizing fragmented and redundant work processes, this solution provides the immediate and longer-term benefits of a Wizard Implementation.
- **Control Center Implementation** is a more comprehensive version of a Wizard implementation that addresses a broader problem domain than is normally appropriate for a Wizard. It provides a similar but deeper set of the same benefits as a Wizard Implementation
- **GES Adoption** provides the full set of benefits offered by the CodiScent toolset and methodology. The services are similar to those of a substantial turnkey or control center implementation but are largely focused on implementing a test case prototype as a training exercise, conducting formal education sessions and mentoring selected customer IT staff.

## Contact CodiScent

Visit our [website](#) to see working demos and learn more about our technology and services, contact CodiScent via [email](#) or contact our principals directly:

### Tel Aviv:

Zeev Chared, Founder and CEO  
CodiScent Ltd.

[www.codiscent.com](http://www.codiscent.com)

Mobile: 011 972 50 752 3070

Email: [zeev@codiscent.com](mailto:zeev@codiscent.com)

### New York:

Howard M. Wiener, Executive Vice President for the Americas  
CodiScent Ltd.

[www.codiscent.com](http://www.codiscent.com)

Office: (914) 723-1406

Mobile: (914) 419-5956

Email: [howard.wiener@codiscent.com](mailto:howard.wiener@codiscent.com)