A Java Framework for Multi-Tier Web-Centric Applications Development

Davide Brugali, Giuseppe Menga, and Franco Guidi-Polanco

Abstract— **Web-centric applications can be understood as middle-size applications where the web container hosts both presentation and business logic. This kind of structure is widespread, because it is adequate when the use of more sophisticated standard component-based solutions (e.g. EJB) introduces unnecessary overheads in development and runtime.**

The aim of this document is to describe our Java framework and tool (G++) which can be used to develop multi-tier web-centric Java applications. This developing infrastructure introduces an architectural reference model that allows design and programming efforts to be concentrated in the development of the specific business logic of the system.

Main support is given to user management, session handling, and request-response cycles. Special support is also offered to database access through our own middleware that acts as a bridge between object-oriented programs and relational databases, hiding major concerns about data storage from programmer.

Index Terms—**Enterprise applications, Business applications, Three-tier architecture, J2EE, Web-centric application.**

I. Introduction

In recent years there has been notable growth in the development of enterprise applications, caused by the confluence of different facts. On the one hand we could mention the end of the service life of many legacy systems, which are no longer able to adequately respond to new business demands. Business managers' objectives and interests also play an important role: they are becoming more aware of the presence of technology as a key factor in a firm's health and competitiveness [1,3]. On the other hand, the evolution of available technology has consolidated certain standards, such as object-oriented languages (Java); multi-tier architectures for large-scale systems; the Internet as a support for ubiquitous networking, protocols for transferring, presenting and exchanges data (HTTP, HTML and XML respectively), and so on. This "de-facto" standardization has rapidly been assumed by the software industry, which continued to integrate most of these elements in the so-called "enterprise middleware" [7]. Enterprise middleware is conceived to build applications that support not just the internal activities of an organization, but also to create the software infrastructure required to go across their boundaries, for example to implement large-scale e-commerce activities.

A Web-centric application is one of the possible configurations that organizes components of an enterprise middleware. This kind of application is widely spread because it results adequate when the use of more complex solutions represent an unnecessary overhead in runtime and development. Typically these applications are divided in three-tiers that groups presentation, logic, and data persistence. They are built around a Web server that controls the user interaction and performs the transactional logic.

However developers, in spite of the adoption of a specific configuration, can be in difficult if they don't handle a clear set of patterns [2] or best practices guiding the application architecture. Regarding this, our recent experience in this field of applications -endorsed by a previous research in patterns, pattern languages, and frameworks for the automation area [11, 12, 13, 14]- allowed us to recognize in Web-centric applications recurrent structural elements and behavioural relationships, which can be encapsulated in a framework of classes for a more efficient development [1]. Specifically for modeling the behavioural dimension we consider the adoption of finite state machines that we already use in automation systems development [4].

In this paper, we describe the implementation of a framework for the development of Web-centric applications for the Java 2 Enterprise Edition (J2EE). This work wants to cover three objectives: first, the definition of an architectural reference, with concrete components and base classes, for developing applications; second, the definition of a set of models to adequately support the application design having finite state machines as basic representation for the application logic; and third, the creation of a tool that allows development of those models, aimed to automatically generate some of the

Davide Brugali is with the School of Engineering of the University of Bergamo, Italy, email brugali@unibg.it. Giuseppe Menga is with the Department of Automatic Control at the Politecnico di Torino, Italy, email menga@polito.it. Franco Guidi-Polanco is a PhD student at the Politecnico di Torino, email franco.guidi@polito.it.

The work presented here has been supported by the Centro di Eccellenza per le Radio Comunicazioni (CERCOM) at the Politecnico di Torino.

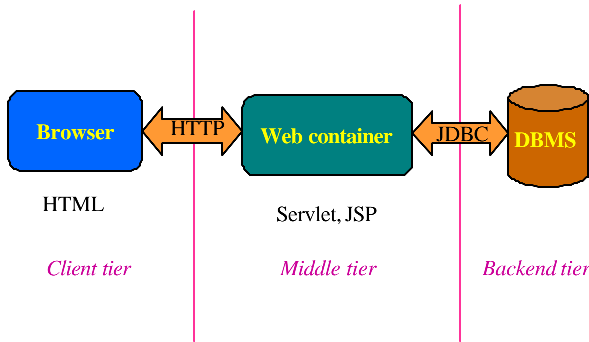

Figure 1: Web-centric application tiers

application code. As result of the integration of these three objectives we obtained an innovative approach and tool for applications development.

This document is structured as follows: Section II presents an overview of the application domain, giving an operational and functional description about the nature of the considered system. Section III describes the model where relies the architecture of the framework. Section IV describes the framework implementation, identifying its components and base classes. Section V presents the tool that we developed for supporting application development. Finally, in Section VII some conclusions about the work are given.

II. The Application Domain

The framework we propose was created for the development of business applications using a subset of the technologies offered by the J2EE. In order to increase the understanding our work, we will first offer a brief description of relevant components of J2EE, and the functional characteristics of a Web-based business application.

A. Overview of the J2EE technology

Sun's J2EE technology is a platform for multi-tier distributed applications. The J2EE architecture basically identifies three tiers in applications, which provide the components and technologies to be adopted at each tier [7]:

1) Client tier

In this tier the means for user/application interaction are given. Standard browsers can support this interaction by using plain HTML pages, dynanic HTML pages generated by JavaServer Pages (JSP) technology, or Java Applets, wich communicate with Web-standard protocols (HTTP, HTML XML). Clients can also interact through a Web-services interface, or even, stand-alone Java client applications, communicating through an HTTP or RMI-IIOP protocols. Exploiting Web-protocols, it is also possible to implement clients in other programming languages (e.g. C++).

Physically, clients can be located outside or inside of corporate firewalls.

2) Middle tier

The middle-tier is the host for the application logic. The application logic is implemented through the definition and interaction of a set of different components, which are loaded and executed in *containers*. Containers are runtime environments, which provide specific component services. They act as an interface between a component and the low-level platform-specific functionality that supports the component. J2EE specifies two types of containers for the middle tier:

- *Enterprise JavaBeans (EJB) container*, which manages the execution of enterprise beans.
- *Web container*, which manages the execution of Servlet and JSP page components.

Enterprise JavaBeans represent a server-side component model for a distributed component transaction monitor [5].

Servlets correspond to a set of Java classes used to extend the capabilities of servers that host applications which can be accessed through a request-response programming model. Although Servlets are conceived to respond to any type of request, they are commonly used to extend applications hosted by Web servers. Specific HTTP-Servlet classes are defined for such applications [7].

JSP pages are text-based which are documents designed to provide a declarative, presentation-centric method of developing Servlets. JSP pages contain two types of code: static template data, which can be expressed in any text-based format (HTML, WML or XML), and JSP elements, which construct dynamic content. Basically, both Servlets and JSP pages describe how to process a client request to create a response [10].

3) Backend tier

Enterprise applications usually have to be able to access applications running on enterprise information systems, which provide the corporate information infrastructure. Examples of enterprise information systems include enterprise resource planning systems (ERP), mainframe transaction processing systems, relational database management systems (RDBMS), and other legacy information systems.

For the backend tier, components derived from the Java Connector Architecture (JCA) are used to access external resources. The most common case corresponds to relational databases accessed through the JDBC connector.

As was presented in the introduction of this article, several configurations of J2EE components and technologies can be conceived to implement a business application. Our framework was developed to cover the configuration represented by Web-centric applications [8], that is, middle-size systems organized in three tiers, but restricted to user interfaces

implemented in standard browser, application logic which is wholly supported by the Servlet technology (no EJB are used). The presentation is created by JSP pages, and data persistence is managed by an external relational database system. Figure 1 shows the components and their interactions in the case of a Web-centric application.

B. Profile of Business Applications

Business applications basically represent systems that hold mission-critical information within a company. They have embedded business rules that reflect the natural structure of the business. They are conceived under the paradigm of improving exchange of information that results in improved business efficiencies and more effective customer interactions.

Today business applications are designed to support many users who have a common business goal and are connected by the enterprise intranet in a Local-Area Network (LAN), or use the Internet when they are outside of the corporate network. Users can perform different activities, and as result of this interaction information is given to the user, and/or the state of the system is updated.

From the point of view of usability, Web-based business applications adopt a graphical user interface that runs in Web browsers, and access to business logic is provided through Web servers. Some business applications are also conceived to support real-time interaction with customers.

III. The Architectural Model for Business Applications

Our framework is conceived to support the implementation of the middle tier of a Web-centric application, where all the logic of the system resides. In this section we offer the architecture of applications developed under our framework paradigm, describing first the model of layers in which application responsibilities are distributed, and then the functions of their different components.

A. Layers

In our architectural model, middle-tier application components are organized into three layers of responsibility, constituting a stack where an underlying layer directly supports the functions offered by a superior layer (see Figure 2). The layers, we identify, are:

- *Interaction layer:* it contains the classes in charge of controlling the interaction with clients. The classes grouped in this layer are responsible for accepting external requests, addressing them to the business logic layer, and presenting the response defined by that layer to the client.
- *Business logic layer*: it contains the classes that encapsulate the transactional logic of the application. Both business rules and entities over the transactional logic acts reside in these classes. The Business logic layer receives requests for transactions from the interaction layer and returns the response to the request to it. This layer interacts with the Data services layer in order to gain access to persistence of business data.
- *Data services layer:* this layer groups those classes used to manage the persistence of the state of entities which belong to the business logic layer in an external database. It also supports preservation of the state of some structural objects managed directly by the Application engine.

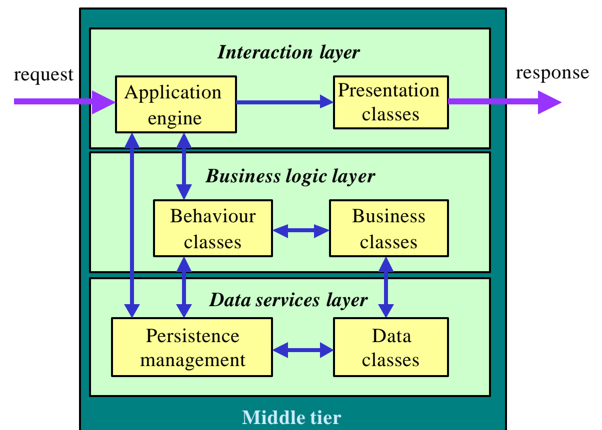

Figure 2. Layers of the middle-tier

B. Components

The architectural model we propose is made up of the following elements:

1) Application engine

It corresponds to the controller component of the application, and resides in the Interaction layer. Its role is to accept requests coming from clients, and decides how to map them into service invocations over specific Behaviour classes in the underlying Business logic layer. This component also manages the preparation of the response that the application should give to the requesting client. Such response is based on the selection of a suitable Presentation class, and yhis choice is made by the Behaviour object invoked to execute the service.

The Application engine is also responsible for storing the state of interactions between different users and the application, in separate *user sessions.* A user session is conceptualized as a macro-thread where transitions between different tasks that compound a behaviour are sequentially performed. Each session has associated a *State object* that encapsulates all the states of each interaction of a single user. This state object enables the reconstruction of the interaction of a given user, in different instances of a Behaviour class, or even by different Application engines. This externalization of

the state increases the scalability of the application and makes it possible to apply load balancing mechanisms through the replication of their classes in different machines, because a session started in one server can be continued in another one. The Data services layer supplies persistence of sessions directly.

2) Presentation classes

These are classes which also belong to the interaction layer, and their function is to creation of an input/output interface. Instances of presentation classes are created and initialized by the Application engine, after a received request has been processed by the corresponding Behaviour class instance.

Interfaces created by presentation classes contain three elements:

- Output data (results)
- Input fields
- Controls (buttons)

Output and Input fields are optional components; controls instead are always present because they allow event generation when there is an user interaction. All of these elements are application specific.

3) Business classes

These are classes which belong to the Business logic layer. Business classes correspond to entities which have a semantic value in the application context, and therefore, they are present in the domain model of the system. They are traditional classes in the sense of object-oriented programming. Examples of Business classes, which can be found in common applications, are: “invoices”, “contracts”, “customers”, “items”, etc.

Business classes are characterized by their attributes that own a persistent state. Persistence is achieved indirectly by the support of *Data objects*, which belong to the data services layer. Instances of Business classes also have links to other instances, so that they can represent the associations, creations or relations of use, that exist in the real system.

Methods of Business classes are grouped in two sets:

- Methods used to access and update the state, and to perform object services within the context of the application domain (e.g. methods set and get attributes).
- Methods used to create and manage instances and their access to the database.

Finally, another characteristic of Business classes is the possibility of having shared accesses to their instances from different user sessions.

4) Behaviour classes

Behaviour classes represent the dynamic aspect of the system, and they are modeled in the Business logic layer. Each Behaviour class corresponds to a logical partition of the entire business application function, as a result of grouping together those activities performed over a common core of Business classes.

Behaviour classes encapsulate *tasks* that correspond to the set of steps required to port a Business object or a set of them, from one meaningful state to another meaningful state in the business domain. Tasks are built by combining the low-level behaviours offered by Business Objects into a single Behaviour class’s method. For example, in a sales application a Behaviour class can implement the “Order management” function and some of the tasks can be “prepare order” or “dispatch order”.

Collaboration relationships can exist between Behaviour classes. This means that tasks which belong to one Behaviour class can require the execution of tasks belonging to other Behaviour classes. For example, the task “dispatch order” pertaining to the “Order management” Behaviour class can invoke the execution of the task “create invoice”, which belongs to the “Invoice management” Behaviour class, before the latter is completed.

In our view, tasks are subsets of the general behavior of the application that can be described by finite state machines. Each state of the finite state machine is in direct correspondence with a presentation class, in other words, each state has an associated Presentation class that provides a client interface. Because states represent phases of user interaction, events necessary to port the system to a new state correspond to a submitted request coming from clients. These requests are the result of user selections regarding the next state to be reached, but they are offered to the user in a comprehensible way (e.g. as option menu).

Behaviour classes don’t have attributes. When a task is executed the state is loaded from an external *Session object*, and after completion, is saved in the same object.

5) Persistence management

The development of object-oriented enterprise applications requires external resources of data to be accessed. However, such resources don’t usually offer an object-oriented high-level interface, as is the common case for relational databases.

Under the Persistence management concept we include a set of base classes that allows general object-oriented-like accesses of application components to an external enterprise data repository.

6) Data classes

Data classes are specific application classes derived and collaborating with the Persistence management subsystem. These classes belong to the *data services layer*, and they are used to support the persistence of attributes of Business classes. Data classes are associated to tables, records or fields depending on the granularity of the data object they represent.

IV. The Framework Implementation

We implement our framework using a subset of J2EE technology. The main component element used is the Web

container in which runs all the transaction logic. This component corresponds to the middle tier in the general application architecture.

1) The `BAFServlet`

The `BAFServlet` class implements the Application Engine component logic directly. It corresponds to a concrete extension of the `javax.servlet.http.HttpServlet` class [10]. This Servlet implements the `doGet` and `doPost` methods used to accept HTTP requests sent by clients. These methods extract parameters from the request containing identifiers (unique in the application context) of the type of event associated to the request, and using this value the `BAFServlet` instantiates the suitable Behavior object responsible for dealing with the request, and invokes the service that carries it out. As a result of this invocation, the `BAFServlet` receives the necessary data from the Behaviour object to provide as a response to the client, referring to a JSP that should be used to present the response.

Some infrastructure services are provided directly by the `BAFservlet` class. The `BAFServlet` encapsulates services for session and user login management, and in these cases no extra Behaviour classes are required.

The implemented user session management (see Section III) provides a flexible mechanism to suspend and resume working sessions, independently of the location of the machine where the session is restored, or where the instance of `BAFServlet` called to deal with the session runs. This enables replication of web-containers and application classes for scalability, fault tolerance and load balancing of the system, because successive states of sessions can be executed in a transparent way by different servers.

The `BAFServlet` is responsible for moving through the different pages that make up an application. The implemented mechanism includes the control of the stack of the presentation pages, allowing the creation of a navigation structure that emulates traditional systems organized around hierarchical menus.

Because the `BAFServlet` is the core component, it is also responsible for initializing and destroying classes which belong to the data services layer, and in particular, those classes which maintain the connection to the database.

2) The `RequestHandler` *class*

This is the base class for the implementation of all Behaviour classes. The `RequestHandler` class offers a single public method, `handleRequest`, which is invoked by the `BAFServlet` when a client request is received. First this method initializes the Behavior class instance with the corresponding user session; afterwards it invokes the methods which the subclasses should implement with the logic of each task.

The `RequestHandler` class implements the forward method used to communicate to the `BAFServlet` the name of the JSP to be displayed as a response to the client request.

3) Java Server Pages (JSP)

In our model presentation is executed through the definition of JSP which build each page offered to the client tier. JSP are selected by Behaviour classes and invoked by the `BAFServlet`, using the "JSP model 2" approach [10].

JSP are application-specific, and in order for them to be implemented the framework offers blocks of scripts to include in each page. The framework also specifies the structure of the requests, in terms of certain parameters and values to be submitted and how they correspond to available events on the displayed page.

4) The "Persit" framework

Persist is a framework we have developed to manage the mapping between objects which belong to object-oriented Java programs and the persistence of their states in relational databases. It is built over the JDBC connector architecture, and offers an objectified high-level interface to data stored in the database. From the architectural point of view, classes from Persist are grouped in two levels:

- *The meta-level*, where the Data Definition Language (DDL) resides. This language allows the database schema to be specified. At this level the conversion between SQL types to specific database vendor types is also specified. The management of object identifiers and their mapping to table keys in the relational schema are also dealt here.
- *The operative level*, which implements the Data Manipulation Language (DML). In this way the application can manage tables, as if they were object collections. It handles records as concrete objects, and fields as their attributes. This enables the programmer to build simple database queries (select, update, delete), as well as sophisticated ones (i.e. selections with nested join of tables). The DML implemented in Persist supports transaction management which guarantee data integrity in complex update operations.

5) The `BusinessObject` *class*

`BusinessObject` is the base class for the implementation of Business classes. Basically the `BusinessObject` class encapsulates the persistence mechanism which keeps the state of instances of Business classes. The `BusinessObject` class enables the Business classes state to be mapped over one or more tables, and it provides mechanisms for restoring the state using simple queries, in the former case, and queries with JOINs, in the latter case. Subclasess of `BusinessObject` inherit methods for instance management.

6) The Session class

Session class instances are used to preserve the state of

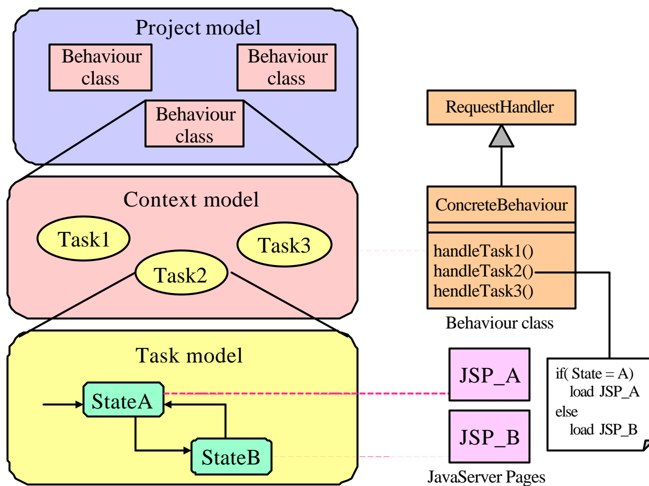

Figure 3: Relationships between models and classes

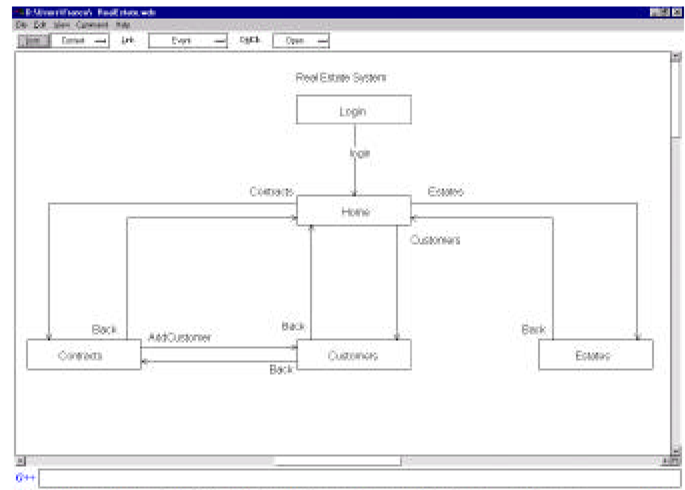

Figure 4: The Project editor

interaction of the user session. The session class provides the `BAFServlet` with methods to update and retrieve the session state, as for example the "valid" or "expired" states.

The session class is implemented as an extension of `BusinessObject` class, allowing it to manage the persistence of the state of the user session.

7) Other classes

The framework includes several minor service classes used to handle errors, to carry out tracing, and other structural internal functions.

V. Supporting Tool

In order to support the development of applications according to our framework, our G++ development environment [4] has been extended in two ways: first, the framework's component and base classes described above has been added to its library of classes; second, and as innovative aspect, a new diagram to model for modeling business logic in web applications, which is called "Interaction Tree Diagram", is introduced.

An interaction tree diagram is conceptually a state transition diagram specialized so that the interactive behavior of a session can be described. In this diagram each elemental state is associated to a presentation class (a single page presented to the user), and transitions between states correspond to events coming from user actions in the client tier (i.e. submitted requests). For convenience of implementation the whole interaction is split up into three hierarchical layers:

The Project layer – representing the whole session and collecting all the Java Behaviour classes of the application;

The Context layer – representing one Behaviour class with its methods;

The Task layer – one method specification of a Behaviour class in terms of an elemental state transition diagram (where states are in association with pages to be displayed to the user).

The interaction tree diagram is conceived so as to approach systems modeling in terms of a top-down functional breakdown (see Figure 3). Each diagram is described below.

A. *Project editor*

The Project editor allows high-level functions recognized in the application domain to be modeled. Each function is represented as a box and oriented arcs represent relationships of access between these functions. Events porting from one function to another function are identified as labels over the arcs.

In all project models the special "`Login`" function is defined, because it corresponds to the entry point for the users who access the system.

Figure 4 shows an example of a project diagram built as a part of a model developed for a real estate system. In the case of this example "`Home`" is the function the system activates when the user logs-in for the first time. The diagram of the example also shows three other boxes, "`Contracts`", "`Customers`", and "`Estates`", which are main functions identified in the system, and in which the business logic of the entire application is distributed.

Each high-level function presented in the Project model can be further described in low-level specifications. At this point the following two possibilities exist: i) the breakdown of the function into the different tasks (using the Context editor) and then, the specification of each task (using the Task editor); ii) the specification of a single JSP associated to that function, in the case in which the function has a single task with a single state.

The transitions indicated in the diagram represent events that allow navigation between functions. It is worth noting, for example, that the event "AccessContract" is used to port the system from the "`Home`" to the "`Contract`" function, and there is a "`Back`" event that makes it possible to return to the

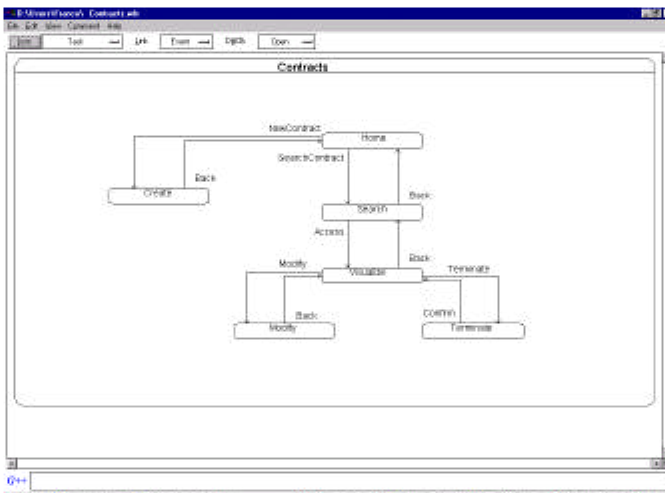

Figure 5: The Context editor

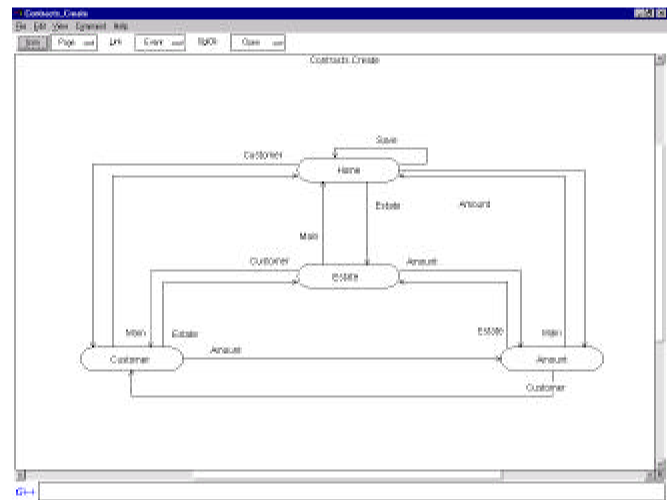

Figure 6: The Task editor

"`Home`" function.

B. Context editor

Each function identified in the Project diagram can be described at a second level of refinement using Context diagrams in the Context editor. A Context diagram offers an internal view of a function, where its tasks are made explicit. In the Context diagram tasks are represented as rounded boxes, and in the same way as in the Project editor, arcs, containing the name of the event that triggers the interaction, represent interactions between tasks.

Figure 5 shows the Context editor window exposing tasks associated to the "`Contracts`" function. This diagram contains a task called "`Home`" that corresponds to the entry point to the function, which is the task that will be performed when the user accesses the context function. The name of the task used as entry-point can be customized in the Context editor for each single function. There are other tasks –Create, Search, Visualize, Modify, Terminate- which contain the logic necessary to create a new Contract, to search searching existing contracts, to display the contents of a single contract (once found) or to modify and terminate those contracts.

The Context editor provides automatic Java code generation of Behavior classes in correspondence to one context diagram of one Behaviour class.

The Context editor is associated to two other editors:

- The *Task editor*, which describes the internal logic of each single task.
- The *Class editor*, which is used to visualize or edit the Java Behaviour classes, generated by the Context model.

C. Task editor

As was described in Section III, each task identified in the Context can be internally described as a finite state machine.

The Task editor, shown in Figure 6, associates each page presented to the user to single states, required to perform the lower level activity. The states are linked between transitions that represent events the system should recognize. Once transitions become specified, the Task editor can be used to generate the code of JSP templates automatically. These templates contain portions of code that implement the logic of navigation, in the form of buttons offered to the user. For example, the JSP code produced for the "`Estate`" state is the following:

```
<%@include file = "../baf/MenuHeader.jsp"%>

  <!-- getButton(classStyleName,eventToSendOnPress,labelOfButton); -->
  <tr><td><%=getButton("ButtonLarge","Amount","Amount")%></td></tr>
  <tr><td><%=getButton("ButtonLarge","Main","Main")%></td></tr>
  <tr><td><%=getButton("ButtonLarge","Customer","Customer")%></td></tr>

<%@include file = "../baf/MenuTailer.jsp"%>
```

Further development of the JSP page requires filling the form with the data and visual controls to present in the page, and this can be accomplished by integrating an external JSP/HTML editor on to the system.

D. Class diagram and class editor

The class diagram editor, and the class editor are the tools the G++ environment supports from earlier versions.

The class diagram offers UML [9] notation so as to model structural relationships between Business classes. The class editor offers a graphical representation of single Java classes, and can be used to edit Business classes and Behaviour classes.

VI. CONCLUSION

Development of Business applications is possible thanks to the existence of enterprise middleware that offers the underlying technology to create middle- and large-size systems. In fact, J2EE offers a set of components that can be

selected to implement configurable solutions according to variable business requirements.

However, developers who use these technologies for the first time should adopt or create it own implementation reference, so as to define the architecture of the application. Using this framework we suggest that developers interested in the creation of web-centric business applications can benefit from structured and reusable know how, which is reflected in patterns of interactions, and a library of components and classes, and can be used to support their work.

In our experience, applying this framework to professional projects, we notice that software productivity is increased and the process of system design and development is speeded up.

REFERENCES

[1] M. Fayad, D. Schmidt, and R. Johnson. Implementing Application Frameworks. New York: 1999.

[2] E. Gamma, R. Helm, R. Johnson, and J. Villisides. Design Patterns. Addison Wesley, 1995.

[3] S. Helal, S. Su, J. Meng, R. Krithivasan, and A. Jagatheesan. "The Internet Enterprise", *in Proc. Symposium on Applications and the Internet, SAINT'02,2002*.

[4] G. Menga, G. Elia, and M. Mancin "*G++: An environment for object oriented design and prototyping of Manufacturing systems*". Intelligent Manufacturing: Programming Environments for CIM. Springer-Verlag Ltd., 1993

[5] R. Monson-Haefel, *Enterprise JavaBeans*. Milano: O'Reilly & Associates, Inc., 2001.

[6] P. Pramongkit, T. Shawyun, "Strategic IT Framework for Modern Enterprise by Using Information Technology Capabilities", IEEE Trans. Neural Networks, vol. 4, pp. 570–578, July 1993.

[7] *The J2EE Tutorial*. Sun Microsystems. [Online] Available: http://java.sun.com/j2ee/tutorial/1_3-fcs/index.html.

[8] *Designing Enterprise Applications with the J2EE Platform*. Sun Microsystems [Online] Available: http://java.sun.com/blueprints/guidelines/designing_enterprise_applications/index.html

[9] OMG, "OMG Unified Modeling Language Specification", Version 1.4, 2001.

[10] T. Loton et al. *Professional Java Servlets 2.3*. Birmingham: Wrox Press Ltd., 2002.

[11] D. Brugali, G. Menga, and A. Aarsten, "The framework lifespan", *Communications of the ACM*, vol 40, no 10, pp. 65-68, Oct 1997.

[12] D. Brugali, G. Menga, and S. Gallaraga, "Intercompany supply chains integration via mobile agents", in *Globalization of manufacturing in the Digital Communication Era of the 21st Century*. Norwell, MA: Kluwer, 1998.

[13] M.E. Fayad, D. Hamu, and D. Brugali, "Enterprise frameworks characteruistics, criteria, and challenges" *Communications of the ACM*, vol 43, no 10, pp. 39-46, Oct 2000.

[14] D. Brugali, G. Menga, "Architectural models for global automation systems", *IEEE Transactions on Robotics and Automation*, vol 18 no. 4 pp 487 –493, Aug 2002.