A Framework for the Efficient Production of Web Applications

Jia Zhang
infiNET Solutions
1425 E. Busch Parkway
Buffalo Grove, IL 60089, USA

Ugo Buy
Computer Science Department
University of Illinois at Chicago
Chicago, IL 60607

Abstract

To date developers can exploit a wealth of existing languages and tools when designing and implementing web applications; however, comprehensive and automated tools and techniques for producing these applications are still conspicuously lacking. Here we describe a framework for efficient production of web applications, which synergistically integrates techniques for automatic code generation with some of the most recent technologies for web development, such as the J2EE application server and XML. We also describe a general-purpose architecture for web applications and our Mockup-Driven Fast-prototyping Methodology (MODFM), which is based on well-tried techniques, such as rapid prototyping and mockup-driven software design. Finally, we discuss an example of a web application developed with our methodology, a comprehensive university administration system that was recently deployed at a research university. Our initial experience with this system indicates that MODFM and related tools can provide dramatic improvements in developer productivity, while also enhancing the reliability of the resulting applications.

1. Introduction

Embracing both computing and networking technologies, web application development can be quite complex, costly and time-consuming, if not supported by a practical methodology [Fraternali00]. To date developers can exploit a wealth of existing languages and standards when designing and implementing web applications. By providing full support for Enterprise JavaBean (EJB) components, Java Servlet API, JavaServer Pages (JSP) and XML technology, the Java 2 platform, Enterprise Edition (J2EE) defines a de facto standard for developing multi-tier interactive web applications [J2ee]. Keeping developers from the tedious coding of system level utilities [Grundy02], an application server is the middle tier of enterprise software linking the back-end systems and databases at one end with the graphical user interface at the opposite end; therefore it is the basis of a typical web application. Most commercial application servers available today are based on the J2EE platform, including Weblogic [Bea], Websphere [Websphere], and JRun [Jrun]. The eXtensible Markup Language (XML) is considered a universal format for structured documents and data definitions on the web [McLaughlin]. Together with the Hyper-Text Transfer Protocol (HTTP), the Hyper-Text Markup Language (HTML) and various scripting languages (e.g., JavaScript), J2EE and XML have enabled the explosion in the number of web applications that we have witnessed to date. However, tools and techniques for developing and connecting the main components of a web application (e.g., interfaces, application logic, and back-end databases) are lacking. As a result, developers often resort to ad-hoc strategies for integrating the various tiers. In all but exceptional cases, developers must design and code web applications from scratch, which wastes valuable human resources and may result in schedule delays.

Here we define a framework for efficient production of web applications. Our framework has several goals. First, we seek to expedite the development of web applications by exploiting automatic program generation techniques. Second, our framework supports the inclusion of customer feedback early in the development process. Finally, we seek to facilitate software maintenance. We accomplish our goals in several ways. First, we define a software architecture suitable for a broad variety of web applications. This architecture seamlessly integrates cutting-edge technologies for web development, such as J2EE and XML, with established development methodologies, including rapid prototyping and automatic code generation. Second, we propose a Mockup-Driven Fast-prototyping Methodology (MODFM) for developing all the components typical of web applications. Utilizing well-tried concepts, such as client-centric development [Sommerville] and rapid prototyping [Szekely], MODFM seeks to gain customer feedback at early stages of development in order to avoid wasting development efforts because of incorrect or incomplete specifications. In particular, MODFM leads to the production of a skeleton system (i.e., the mockup) that contains the interface, but not the full functionality, of the finished application. Finally, the use of automatic code generation can provide dramatic improvements in development speed. We developed a university administration information system using MODFM. Our empirical

Proceedings of the Eighth IEEE International Symposium on Computers and Communication (ISCC'03)
1530-1346/03 $17.00 © 2003 IEEE

experience shows that MODFM and automatic code generation can provide dramatic improvements to software productivity.

The rest of this paper is organized as follows. In Section 2, we discuss related work in web development. In Section 3, we introduce a general-purpose software architecture for web applications. In Section 4, we describe our web code generator. In Section 5, we present our framework for web applications. In Section 6, we discuss MODFM and the university application example. Conclusions and future work directions are discussed in Section 7.

2. Related Work

A lot of research work has been conducted to simplify and automate the development of web applications. Among various efforts, HDM [Garzotto93] defines a popular model derived from the Entity-Relationship Model [Chen76] for hypermedia application design, which divides conceptual schema into two categories: structural and navigational. Autoweb utilizes a variant notation of HDM called HDM-lite, which adds a presentation schema to the conceptual design of web applications. The conceptual schemas are stored along with the data content in the development database [Fraternali00]. Jweb provides a design and prototyping environment that integrates XML technology with HDM to help design the conceptual schema [Bochicchio00]. RMM is a database-driven methodology for structured hypermedia design. Its main idea is to provide a visual representation of the system in order to facilitate design discussions [Isakowitz95]. RMM defines an iterative process to refine and transform visual components into database elements. Gaedke uses WebComposition and the WebComposition Markup Language (WCML) to present a systematic approach for code reuse in component-based web applications [8,9]. IIPS models navigational structure, compositional structure, and user interface through ontologies; it also provides tools for code generation [Lei02]. Struts provides an open source unified front-end MVC framework [Gamma94] for web applications [Struts]. The Struts framework has three main components: a servlet controller that is provided by Struts itself, JSP pages (the view), and the application's business logic (the model). The servlet controller delegates HTTP requests to appropriate handlers (actions).

All those efforts defined techniques and tools to speed up design and development of web applications. At this moment, however, it appears that all existing techniques address only specific aspects in the development of web applications. An additional limitation is that those methods may or may not integrate easily with current web technologies. Here we seek to provide efficient support for the end-to-end process of generating and maintaining whole web applications. In contrast with previous approaches, we accomplish this objective by providing a generic framework to support generating a running mockup application and by using automatic code generation techniques that exploit state-of-the-art tools and technologies for web development.

3. Two-tier MVC Architecture for Web Applications

We propose a two-tier MVC architecture for web applications that uses the Struts server [Struts] for the front-end architecture, as shown in Figure 1. A web application system is divided into a front-end tier and a back-end tier; each tier is organized according to the MVC paradigm. We chose this paradigm to decouple the data model and application logic from the user interfaces. The front-end tier includes JSP pages, servlets and the Struts engine, while the back-end tier comprises all EJB engines and the database. Compared to the usual three-tier architecture that is quite popular in web applications [13,24], our front-end tier can be considered as a normal front-end tier, while our back-end tier contains the normal middle tier and back-end tier. We choose this architecture because the most recent middle-tier tools (e.g., EJB) effectively incorporate the database while hiding the details of database organization from the front-end tier [Szekely].

In the front-end tier, we formalize the Struts framework as follows. Every JSP page represents a view, together with a form bean holding the contents of the

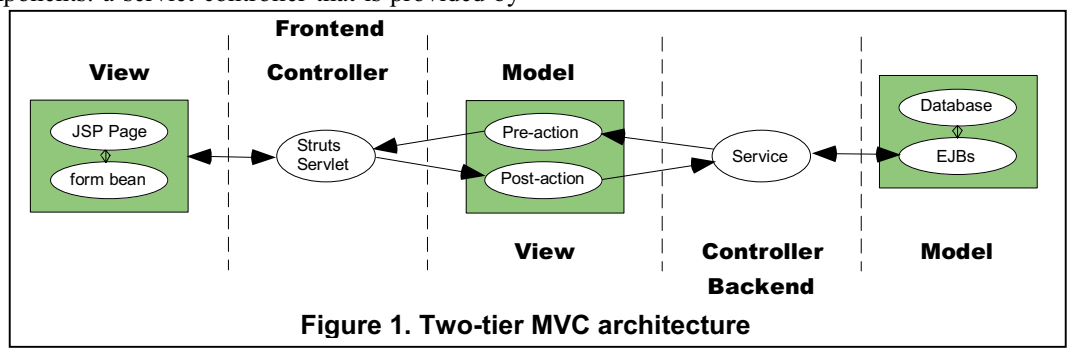

Figure 1. Two-tier MVC architecture

page. A servlet inheriting from the Struts servlet acts as the controller. Two action classes–a (pre-, post-) action pair–act as the model. Pre-action prepares the contents for the JSP view, while post- action gathers user input from the JSP page and performs some operations (e.g., error checking, formatting). The advantage of this architecture is that it captures the essential scheme underlying each web page. In this manner, the flow between a user and the system is clear. For the back-end tier, EJB implements the application model and hides details of database organization. We construct a service layer to be the controller, which deals with all application logic. Actions act as the view component, as well as the model of the front-end tier. The rationale is that the purpose of the back-end tier is to provide system state information to the front end. Thus, we can consider actions as agents providing different ways to present back-end states to the front-end tier.

This architecture clearly identifies an object-oriented, component-layered structure for web applications. In addition, it leads to an ideal organization of all of the code packages in a web system. According to this architecture, any module in a web application can be realized by the composition of JSP pages, form beans, pre-actions, post-actions, service methods, EJB components, and database schemas.

4. Web Code Generator

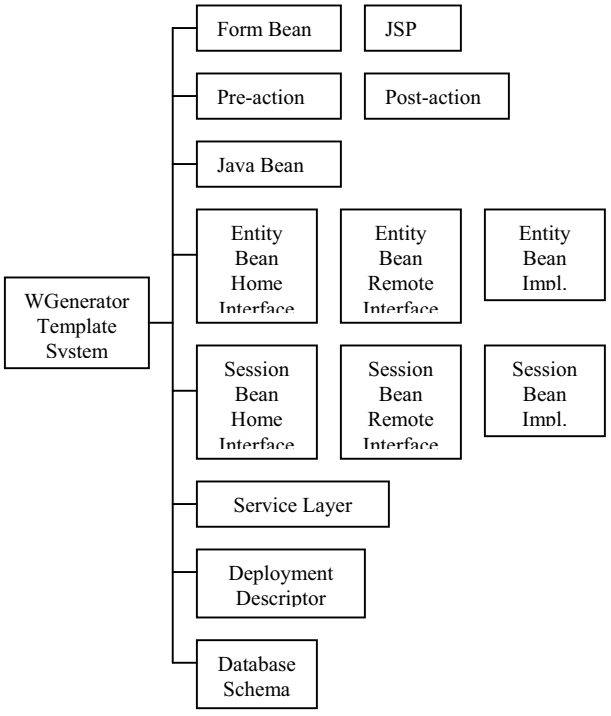

Figure 2. WGenerator template system

We designed and implemented a web code generator called WGenerator. Similar to existing code generators [5,12,15,16], WGenerator uses predefined templates to generate various kinds of code units. The following features characterize WGenerator. First, WGenerator conforms to the underlying software architecture that we described in the previous section. Second, based on this architecture, WGenerator provides a complete set of templates that can generate a fully functioning running system. Third, the template system in WGenerator decouples the data model from configuration information about a web application. This separation makes it especially easy to read and modify the generation scripts when creating a new version of the application. Fourth, the template system is highly reusable on different web application servers. Fifth, WGenerator integrates utilizes state-of-the-art XML technology.

Following the software architecture discussed in Section 3, our template system comprises 14 templates for various components of web applications. As illustrated in Figure 2, this template system provides code templates from the front-end to the back-end of a web system. JSP and form bean contain templates for JSP pages and their associated data models. Pre-action and Post-action templates define actions to format a web page and to collect information interactively supplied by page users. The Java bean template defines the business data object. We use the entity-bean home interface template, remote interface template, and implementation template to generate EJB entity beans. Likewise, the session-bean home interface template, remote interface template, and implementation template generate EJB session beans. The deployment descriptor template generates deployment descriptor segments for the corresponding entity beans and session beans. The service layer template generates service method signatures. The database schema template generates SQL statements for creating and deleting tables in the database. Given the large amount of code that we generate automatically, a developer merely needs to define the data model and to implement the business logic in order for the web application to be complete.

The data model should be contained in a list of XML

```
<WGenerator>
 <OBJ NAME="StudentAddress" TNAME="student_address">
  <ATTR ATNAME="StudentId" ATYPE="String" VALID="YES"/>
  <ATTR ATNAME="Address" ATYPE="String" UNIQ="NO"/>
   <ATTR ATNAME="City" ATYPE="String" UNIQ="NO"/>
  <ATTR ATNAME="State" ATYPE="String" UNIQ="NO"/>
  <ATTR ATNAME="Country" ATYPE="String" UNIQ="NO"/>
  <ATTR ATNAME="ZipCode" ATYPE="String" UNIQ="NO"/>
 </OBJ>
</WGenerator>
```

Figure 3. A simple example of a data file

files. Figure 3 is an example of a data file that defines a data model of the *StudentAddres* object. As shown in Figure 3, for every field of the corresponding data model, one can define its name and data type. *StudentAddress* contains ten fields: student id, address, city, state, country, and zip code. The *Key* field is used to store the primary key internally. Every data file is applied to template files when generating the corresponding code.

One can also define a configuration file to specify generation criteria to WGenerator in XML format. We separate the configuration files from the data files in order to improve the reusability of the data files. Figure 4 is an example of a configuration file for the corresponding data model *StudentAddress*. A configuration file defines the relationships between template files and data files. This kind of relationship is a many-to-many relationship. As shown in Figure 4, the data file *StudentAddress* will be used to generate the business object file and entity bean files, among others. In addition, developers can decide whether the target file will be completely regenerated, or partly replaced, by specifying the value of attribute *Change* to be *All* or *Part* respectively, as the attribute of every item shown in Figure 4. Since changes of some data models in the process of design and development are normally inevitable, and this feature can guarantee the flexibility of generated code.

```
<WGenerator outputDir ="univ" data="studentAddress">
 <CODE NAME="StudentAddress">
   <template name="=OBJ=NAME=Bean.java" change="ALL"/>
   <template name="=OBJ=NAME=EJB.java" change="ALL"/>
   <template name="=OBJ=NAME=EJBHome.java" change="ALL"/>
   <template name="=OBJ=NAME=EJBImpl.java" change="ALL"/>
   <template name="ApplicationResources.properties" change="PART"/>
 </CODE>
</WGenerator>
```

Figure 4. Example of a configuration file

5. Software Framework

Based on the two-tier MVC architecture, we constructed MODFMEnv, a framework to support the design and prototyping of web applications. The overall picture of MODFMEnv is shown in Figure 5. MODFMEnv is built on top of an application server; it specifically adopts J2EE technology, such as EJB, JSP, and servlets. Two additional techniques underlying the framework are Struts and the database. The two-tier software architecture that we discussed earlier is the backbone of the framework. Two code generators support the code generation based on this architecture. A menu generator automatically builds code for defining hierarchical menu systems; this component also provides an easy way to associate web pages with the

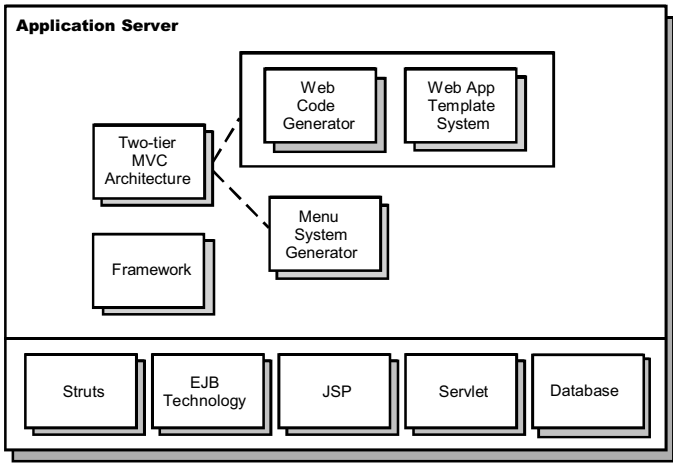

Figure 5. Framework for efficient production

corresponding menu items. A specific application-server oriented template system defines the skeleton of the sets of files to be generated. Our WGenerator, combined with this template system, can be used to automatically generate the whole structure of the system except for application algorithms. Most of the templates in the template system can be fully reused if another application server is adopted; normally only the EJB deployment descriptor template needs to be replaced according to the new server. Another important component is our framework package. This component maintains the relationships between different web pages throughout the menu system. The framework package also provides authentication control, security control, and other utility functions.

6. The MODFM Methodology

Underlying our framework is the MOckup-Driven Fast-prototyping Methodology (MODFM) for the development of web applications. Our methodology employs existing and well-tried techniques: Rapid prototyping, code generation, and mockup-driven software design. Rapid prototyping assures that developers acquire critical knowledge required to build a full system early on. Automatic code generation can enhance developer productivity, improve software quality, and shorten the development cycle. Building mockups as system prototypes has several advantages. First, mockups help elicit and finalize the system requirements. Second, clients can view the layout of the final system at early stages. Third, since a mockup will become the skeleton of the final, no development work is wasted. Fourth, mockups can be reused by other similar applications since they do not involve coding application logic. Fifth, constructing mockups blends functional decomposition [Wieringa98] with object-oriented design

1. Define hierarchical menus.
2. Design web pages for each menu item.
3. Design data model for each page.
4. Call WGenerator to generate mockup system.
5. Map pages with menu items to configure mockup system.
6. Gather user feedback and go back to step 3 for updates.
7. Add business logic to service layer for a running system.

Figure 6. Mockup-driven Fast-prototyping methodology

principles.

The MODFM methodology is summarized in the seven steps appearing in Figure 6. In the sequel, we discuss the various steps of MODFM.

Step 1. In this step domain experts and software designers first specify the requirements and use-case scenarios of the application. Next, designers use functional decomposition to divide the system into modules and sub-modules. Finally, designers organize the modules into hierarchical menus. When all the menu definitions are complete, WGenerator automatically generates Java code that displays the hierarchy of menus. This preliminary prototype is submitted to the customers for review and possibly revised.

Step 2. In this step domain experts design web pages for each menu item identified earlier.

Step 3. In this step designers define the data models underlying the above web pages. In general, these pages can provide intuitive guidelines for identifying the data models. It is natural to use a traditional OO methodology when defining the data models, because this activity is similar to identifying real-world entities during class-oriented decomposition.

Step 4. In this step WGenerator generates the code associated with each page defined previously. The following files are generated: JSP pages, form beans, pre-actions and post-actions, maps in Struts-config.xml, entity beans, deployment descriptors, session beans, service methods, and database schemas.

Step 5. In this step, developers associate JSP pages with the corresponding menu items. This is accomplished quite simply by registering the pre-actions of the page to the corresponding menu item in an XML configuration file. The menu generator and the framework package will handle the full system configuration. At this point, we can get a running mockup system, which has all the user interfaces that will appear in the final system. This mockup system is used to elicit additional feedback from the customers.

Step 6. The purpose of this step is to incorporate customer feedback into the mockup system. To this end, we repeat steps 3 through 5 above. These iterations can be done easily, since all of the programs can be regenerated automatically based on the updated data models contained in the XML files.

Step 7. After the mockup is accepted, developers code the application logic into the corresponding service methods. Real programs replace the initial dummy methods that were automatically generated for the mockup. Before the development of these programs, the clients must approve documentation and project solution packages, in order to ensure the correctness of the application scenarios to be implemented. After this step is finished, a complete system will be ready for testing and deployment.

The first three steps of MODFM require domain experts to analyze the system requirements and to decompose an application. Therefore, these steps do not necessarily involve developers. The fourth step is fully automated. The fifth step (i.e., mapping pages with menu items) also requires domain experts or system designers. The sixth step requires communication between clients and system designers. Only the seventh step, which embeds application logic into the system, requires software programmers.

We utilized MODFM and the MODFMEnv framework to design and develop an e-University suite, which is a web application that serving for students, faculty members, staff members, and administrators for such services as admissions, student records, registration, and financial services. After building up the whole system, we collected statistical data on the final software system by the measurement of lines of code (LOC). Other than configuration files, common files that could be reused for all applications, and JSP files, all of the code can be organized in packages as shown in Figure 7. Although WGenerator helps to generate skeletons of JSP pages from data files, "visual designers" are needed to adjust the display of the pages. Therefore to simplify our statistics, we do not consider JSP package. As illustrated in Figure 7, the files are packages in six directories. All of the code under three directories is completely generated: entity bean, bean, forms, and META-INF. For actions directory, about 59% of the code were generated, and

Directories	LOC	MANUAL	GENERATED
META-INF	17193	0	17193
entity bean	22320	0	22320
session bean	5396	4416	980
bean	14224	0	14224
actions	7916	3255	4661
forms	12695	0	12695
Total	**79744**	**7671**	**72073**
%		**9.619533**	**90.3804675**

Figure 7. Statistical data of the e-University

41% of the code manage exchanging data information with back-end system therefore were manually coded on the basis of generated skeletons. As a result, for altogether 79,744 lines of code, 9.62% of the final code was eventually coded by developers, and 90.38% of the system was automatically generated. Furthermore, it is easy to be observed that as the scale of the web application grows, the complexity of data models will increase dramatically. Accordingly the amount of automatically generated code in relative directories, as shown in Figure 7, will also expand spectacularly. Our initial experience with this system indicates that WGenrator can provide improvements in development speed while also enhancing the reliability of the resulting applications.

7. Assessments, Contributions, and Future Work

Our experience validates the efficiency and effectiveness of our framework and MODFM in developing web applications. We used MODFM to design and develop the e-University system. Originally the system was planned to require six months with four developers. Using MODFM, the first author implemented the system in three months, together with a JSP designer to define the page displays. Furthermore, because the system generated is highly reusable, currently we are ready to modify the system for other universities.

We understand that MODFM requires discipline in design and development. However, in our experience, the benefits of MODFM largely justify this additional effort. MODFM is an architecture-based methodology that leads developers to construct quickly a prototype of the web application; the analysis phase and the development phase are clearly decoupled. The availability of the mockup system gives customers the opportunity to participate in the system design at early stages of the development. WGenerator helps build a running mockup system with little developer intervention. Furthermore, the mockup system can be easily regenerated based on user feedback.

All the application logic is reflected in methods in the service layer. All other layers are automatically generated from templates and data models. The template system can also be reused in other web applications by modifying the data models. Methods in the application logic are likely to be reused as well. Thus, our framework not only supports rapid prototyping; it also provides a highly reusable development environment.

We would like to continue our research work by constructing an Interactive Development Environment (IDE) to support MODFM. Currently developers need to set up data models and configurations as a set of XML files. An IDE can automatically support the definition and maintenance of these files.

8. References

[Bochicchio00] M. Bochicchio and R. Palano, "Prototyping Web Applications", Proceedings of the 2000 ACM Symposium on Applied Computing, 2000, Como, Italy, pp. 978-983.

[Chen76] P.P.Chen, "The Entity-Relationship Model: Toward a Unified View of Data", *ACM TODS,* 1, 1, 1976, pp. 9-36.

[Ejbgen] http://www.beust.com/cedric/ejbgen/#introduction.

[Fraternali00] P. Fraternali and P. Paolini, "Model-driven Development of Web Applications: the AutoWeb System", *ACM Transactions on Information Systems (TOIS)*, Vol. 18, Issue 4, Oct. 2000, pp. 323-382.

[Gamma94] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns*, Addison Wesley 1994.

[Garzotto93] F. Garzotto, P. Paolini, and D. Schwabe, "HDM – A Model Based Approach to Hypermedia Application Design", *ACM Transactions on Information Systems*, 11, 1, Jan. 1993, pp. 1-26.

[Grundy02] J. Grundy, S. Newby, T. Whitmore, and P. Grundeman, "Extending a Persistent Object Framework to Enhance Enterprise Application Server Performance", Proceedings of the 13th Australasian Conference on Database Technologies, 2002, Melbourne, Victoria, Australia, pp. 57-64.

[Isakowitz95] T. Isakowitz, A. Stohr, and E. Balasubramanian, "RMM: A Methodology for Structured Hypermedia Design", *Comm. ACM,* 38, 8, Aug. 1995, pp. 34-44.

[Jrun] http://www.macromedia.com/software/jrun.

[J2ee] http://java.sun.com/j2ee.

[Lei02] Y. Lei, E. Motta, and J. Domingue, "IIPS: An Intelligent Information Presentation System", Proceedings of the 7th International Conference on Intelligent User Interfaces", 2002, San Francisco, CA, USA, pp. 200-201.

[McLaughlin] .B. McLaughlin and M. Loukides, *Java and XML*, O'Reilly Java Tools.

[Sommerville] I. Sommerville, *Software Engineering*, Addison-Wesley Pub Co, 6th edition.

[Struts] http://jakarta.apache.org/Struts.

[Szekely] P. Szekely, *User Interface Prototyping: Tools and Techniques*, 1994, USC/Information Sciences Institute.

[Bea] http://e-docs.bea.com/.

[Websphere] http://www3.ibm.com/software/webservers/sppserv.

[Wieringa98] R. Wieringa, "A Survey of Structured and Object-Oriented Specification Methods and Techniques", *ACM Computing Surveys*, Dec. 1998, pp. 459-527.