A Combined Runtime Environment and Web-Based Development Environment for Web Application Engineering

Martijn van Berkum[1], Sjaak Brinkkemper[2], and Arthur Meyer[1]

[1] <GX>, Toernooiveld 225, Nijmegen, The Netherlands
{mvberkum,arthurm}@gx.nl
http://www.gx.nl/
[2] Institute of Information and Computing Sciences
Utrecht University, The Netherlands
s.brinkkemper@cs.uu.nl
http://www.cs.uu.nl/people/sjaak

Abstract. In the last 5 years, server side application server technology became very popular. The two most popular, .NET and J2EE have a large mindshare in the commercial world. Application servers have raised the productivity of server side web developers. In this paper we give a short overview of the history of application servers, and try to answer the question: what options or features would raise the productivity even more? While building commercial websites at <GX>, we found out that that a combined web-based development environment and persistent runtime environment raised our productivity, even compared to .NET or J2EE environments. In this paper we describe such a system built at <GX>, which we called a WRDE (Web-based Runtime and Development Environment), and show the strength and weaknesses of this approach.

1 Introduction

The evolution of systems to build websites can be viewed in a few steps. In the first step, web servers were very simple and followed a simple client/server model. In the second step, more complex operations were done by spawning an external process for every request, and the connection between this external process and the web server was standardized. The third step was to enable generic components inside the web server itself. The fourth step was making use of this generic component to forward requests to an external application server. The next section describes those steps in more depth and will give some drawbacks of the current existing application servers. We propose some new features which would make server side development easier and more productive.

1.1 Evolution to Web Application Servers

During the first days when the Internet, and especially the web, became mainstream, web servers were seen as a new and simple kind of servers, just like mail servers and

A. Persson and J. Stirna (Eds.): CAiSE 2004, LNCS 3084, pp. 307–321, 2004.

ftp servers. These systems did not need a lot of customization, the only thing they needed to do was deliver files to a client, which then rendered it to show it in a browser. The development environment for websites was therefore quit simple; a text editor to create HTML files and an ftp server to put those files on the server where the web server resided.

After a while, people needed ways to create more dynamic page, for example to place the current time on a page. At first this was done by changing the web server itself, but soon after that a standardized way to create dynamic pages was invented; CGI-BIN, which stands for Common Gateway Interface. For every request for a dynamic page, the web server just starts up a new process on the server. The web server returned the output of this process to the client.

This idea has a drawback; for every dynamic request the web server has to spawn a process, which takes a lot of CPU power. A better way is to add a component inside the web server itself, which is running al the time. There is no generic name for this, in the Apache web server it is called a module, in Microsoft IIS an 'ISAPI' (Internet Server API) and in the Netscape web server it is called NSAPI (NetScape API). Developers can create a module (most of the time written in C or C++, using common development environments) that will run inside the web server process.

The disadvantage of this approach was that when the module crashed, the whole web server crashed. To solve this problem, a new architecture was invented. In this architecture, most of the dynamic web application logic is placed in a separate process, and a small module inside the web server acts as a proxy and forwards any (dynamic) requests to the other process. This has several advantages; the process is more loosely coupled with the web server, and can be implemented several times, to spread the load. Such an application is called an application server, a separate process from the web server. Developers that want to enhance the web server create an application inside those application servers. Common characteristics of these environments are:

- Used for dynamic website
- Tools for deployment, security, session managers are included
- Tools for performance, like automated load balancing and persistent session are part of the environment
- The main purpose is generating ASCII/Unicode, most of the time HTML of XML
- The target for an application server is a web browser or any other HTTP compliant client
- Build for usage in HTTP sessions

The two most popular and competing application server standards are .NET and J2EE. During the last five years both application server standards evolved a lot. Important components in these application servers are:

- Code is running inside a virtual machine. This virtual machine is called a Common Language Runtime in .NET and the Java Virtual Machine in Java.
- A template engine to generate HTML or XML easily, and a visual environment to generate web pages on a higher level (Java Server Pages and Java Server Faces in Java, ASP.NET and Web Forms in .NET).
- A persistent layer which makes the translation between relational databases and the objects in the application environment itself (Enterprise Java Beans in J2EE, not yet standardized in .NET).

These kind of application server also has a lot of tools and libraries to help the developer, for example to parse XML, transaction management tools, mail libraries, tools to send and retrieve messages and build message queues and libraries to send Internet e-mail. See the references for URL's that give a broader overview of these technologies [1][2].

1.2 Drawbacks of Current Web Application Servers

Server-side application servers are powerful. Despite that power, based on the experience of building commercial websites we built during the last 5 years some drawbacks of these application servers became apparent. All drawbacks have immediate impact on development productivity. We have identified the following four drawbacks:

1. *Lack of transparent persistency*. The environment is object oriented, yet objects created inside a virtual machine are never persistent, which means that every object has to be created again when restarting the virtual machine. The problem with this is that the developer has to come up with his own persistence engine to store the objects. Ideally, the developer should not be doing anything to gain persistency, and the underlying virtual machine takes care of the persistence of objects. Such a persistency is also called orthogonal persistency [9]. For larger data-objects some facilities exists, like Enterprise JavaBeans in J2EE, but they are never transparent and a developer has to do lot of administrative work to take care of the persistence, like creating the right configuration files in XML or keeping in mind some of the constraints of the custom persistence layer.
2. *Minimal redeployment is not a design goal*. When a change has to be made in a system, a redeployment of the application inside the application server is needed, which means that the application is not reachable for a short time. This also means that the code-run-test-debug cycle of a developer is taking a lot of time. Changing anything at runtime is hard and gives a lot of complications. An application server should have as a design goal minimization of redeployments, so a developer can focus on solving problems, not on redeployments.
3. *No testing during deployment*. Although you can set up a staging environment to test new code, it is never really possible to simulate the real environment, with thousands of visitors and all kinds of connections with backend systems. The proof is in the pudding with high-traffic websites. Current application servers do not support debugging, testing and changing code at runtime very well, not only because of the application server itself, but also because the underlying language and virtual machine does not support it.
4. *Separated development and runtime environment*. The development environment itself is a separate application, almost always client based. The runtime environment and the development environment are strictly separated. While this can be a good thing, during testing it is very easy for developers to work inside the runtime environment to see the effect of changes immediately. By separating the environments, for every change in the source code a redeployment of the compiled code is needed, which can take a lot of time. To compensate for this, several of the J2EE and .NET development environments like Eclipse and Visual Studio .NET include

an application server or servlet engine, making it possible to test the application in much the same way that one uses a traditional development environment for C++ or Java programs. But even so, these are still two separate environments.

While the current server-side application server systems are very useful and used a lot in commercial and non-commercial environments, we state that they miss three essential features, which makes developing in a web environment much more productive:

1. A runtime environment that takes care of the persistence of objects in the system, completely transparent for the developer
2. The possibility to change everything on runtime, even object type definitions and source code.
3. The development environment is completely web based, which means everything can be done using a web browser.

Such a system can be called a Web-based Runtime and Development Environment (WRDE).

1.3 Related Work in Web Application Engineering

Given the current emerging role of web applications in e-commerce worldwide, there is an abundance of research activity going on in this area. Among others, two main streams can be identified that are relevant to our work: web-service oriented computing, and design of web applications. Based on the W3C and OASIS standardization of web-service technology (e.g. UDDI, WSDL, SOAP, ebXML, WSRP) numerous approaches have been proposed for the composition of complex business applications based on generated or created web-services (see for example Papazoglou, 2003 [13], and Fraterneli, 1999 [14]). Goal oriented design of web applications was suggested in (Liu, 2002 [16]), where a variant of Use Case Maps were combined with a Goal-oriented Requirements Language to model the scenarios of interaction with a web application. In (Garcia, 2002 [17]) an agent based architecture for an e-commerce site was presented to facilitate intermediary operation in complex inter-enterprise business transactions. A powerful modeling language for web sites, called WebML, has been developed by the group of Ceri and Fraternali of the Politecnico di Milano. The accompanying WebRatio case tool enables specifying and generating complete web applications (Ceri, 2002 [15]).

1.4 Overview of the Paper

In the following chapter we will explain the SiteWorks system, in which all of the above characteristics are realized. After explaining the actual implementation of such a system, we will go more deeply into the three characteristic features in chapter 3. In chapter 4 we will give examples of usages of this system in a real world environment. Chapter 5 will give a conclusion and further research possibilities.

2 SiteWorks

2.1 Explanation and Infrastructure of SiteWorks

SiteWorks is built on top of the J2EE architecture, but is itself a virtual machine in which applications are running. It needs an external relational database to store its data in. See figure 1 for the infrastructure SiteWorks needs to get it running.

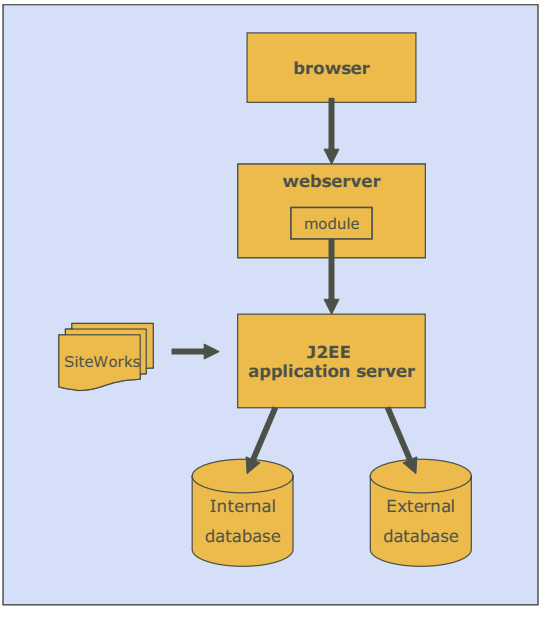

Fig. 1. The SiteWorks infrastructure

The browser is always the interface for the user of the system. When a user requests a page, the browser makes a request to the web server. The web server checks if the page asked for is a special page. If so, the request is forwarded to a module inside the web server (depending on the type of web server, it is called ISAPI, NSAPI or module). This module is very lightweight and simple forwards the request to a separate servlet engine. Servlets are a part of the J2EE specification, and as such, the servlet engine is a part of a J2EE compliant application server. The servlet engine is running inside a Java Virtual Machine and is waiting for requests.

The servlet engine is an application server, and SiteWorks is one of the applications running inside the server. To store internal data, SiteWorks uses an internal database, which can be a SQL 92 compliant relational database. SiteWorks can also connect to other databases to integrate that data in the web solution. SiteWorks also depends on a few third party libraries like JDBC database drivers or XML parsers.

2.2 SiteWorks Internal Architecture

See figure 2 for the SiteWorks architecture. SiteWorks consists of the following components:

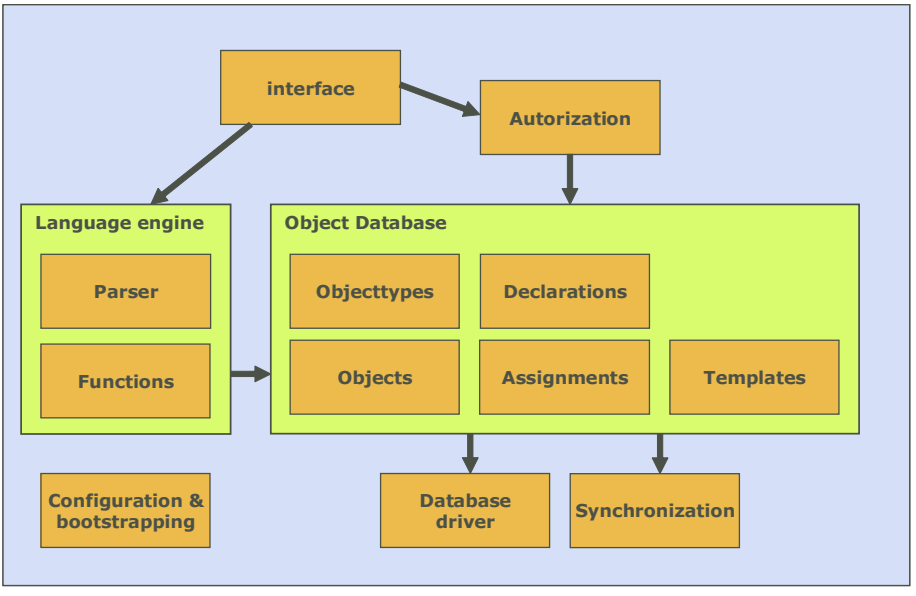

Fig. 2. The SiteWorks architecture

The **configuration and bootstrapping component** takes care of the configuration needed to boot up the SiteWorks system. All other components can retrieve configuration items from this module. As much options as possible are changeable at runtime, as that was one of the requirements of the system. Different than regular J2EE and .NET application servers, a restart is not necessarily needed when the configuration is changed. Also, in this module the necessary bootstrapping procedures are executed. As configuration is partly stored in objects which are not retrieved yet, this is a crucial part.

SiteWorks has a **generic interface** that has two functions; it is a standardized web based user interface for developers (see figure 3) and it is a controller for incoming request. In the user interface for the developer he or she can browse and change the object type definitions, the object population, templates, connections with the database(s), configuration options, user and groups, permissions and maintenance options. All user interface interaction is based on web technology. Developers use a web browser to access and change the system. This interface is also available on live, deployed environments. The controller functionality takes care of incoming requests, for example by checking authorization levels in the authorization module and forwarding the request to the parser.

The **authorization layer** in the SiteWorks system has 3 concepts for authorization and security: users, groups and permissions. Users can be in one or more groups. Groups have permissions. Depending on these permissions users in a group can change or browse through certain object types and have access right to the development environment. The users and groups can also be used when making new web applications with SiteWorks, as users and groups are objects itself, and can be manipulated as any other object.

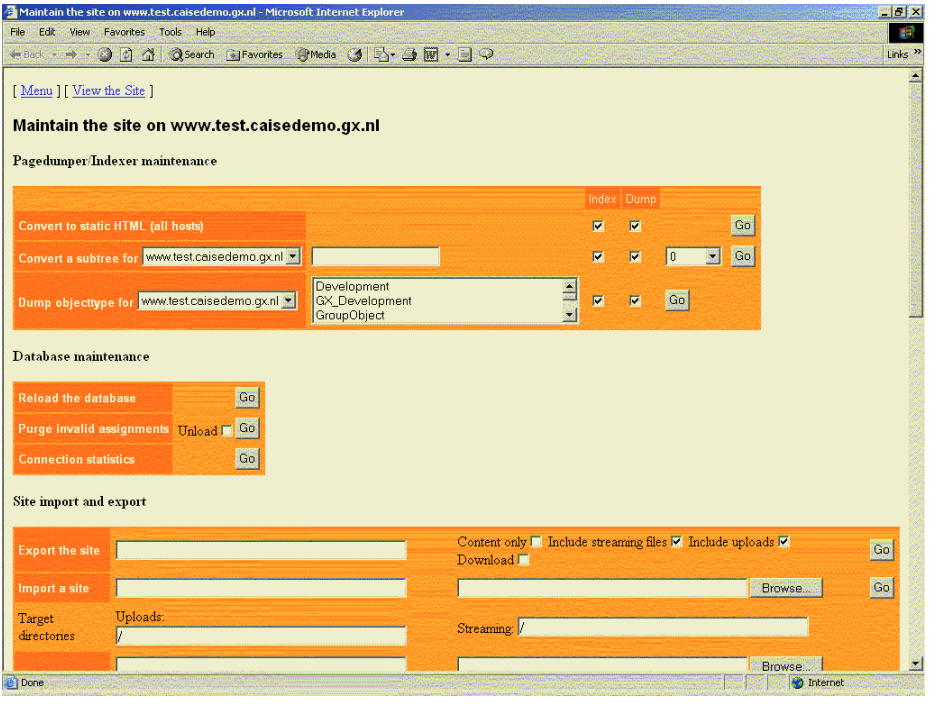

Fig. 3. The SiteWorks user interface

The **object database** is the core of SiteWorks. Five main entities represent this core:

- Object Types are the equivalent of classes in object oriented languages. They have a unique name as an identifier, and have 1 or more declarations.
- Declarations are the attributes of an object type. The combination object type with the name of a declaration is unique. A declaration has at least a name, a type and a constraint (1, 1 or more, uniqueness etc).
- Objects are instances of an object type. Objects are identified by a unique number.
- Assignments always belong to an object and are the filled-in declarations of the object type it belongs to. The combination of object, declaration and order number makes them unique.
- Templates are presentation and logical templates to present and manipulate the objects. A template always has a name. The combination of the template with an object type is unique, therefore the name of the template needs to be unique for all templates of that object type.

For example, in SiteWorks you can create (by configuring) an Object Type ‘page’ with a declaration title and another declaration paragraph. After creating it, you can create instances of this object type, which contain the actual title and text of that page itself. Next to it you can create templates, for example a ‘show’ template to present the page to visitors, and an ‘edit’ template to change the page.

Persistence and storage of the object database in the relational database is done automatically and transparent for the developer. The five entities can be introspected, changed, created and deleted on runtime, using the web interface.

The **script engine** consists of two parts; a parser and a library. The parser parses the templates belonging to the object types and executes it. The language used is arbitrary; in SiteWorks a custom language is used. In theory, it is possible to use almost any object oriented language or object oriented capable language, especially scripting languages, as they need to be executed and compiled on runtime. The language needs to be object oriented, because the language needs to be able to manipulate the object database. A good example of such a language would be JavaScript. The library contains all kinds of functions available for the developer, for example file manipulation or external database access.

To store the data from the object database and to access other databases, SiteWorks has a **database connectivity** layer, which makes it possible to use different kind of relational databases. This module is used by the object database for storage of the objects, and by the library for external database access. The database connectivity layer also contains functionality like connection pooling. The configuration of the database connection can be changed on runtime. For performance reasons, all data of the object database is cached in memory, and only when changing objects or object types, data is written to the database. This also means that access to the internal database for other applications is prohibited, as there is no way to know for SiteWorks when the database is changed.

Web applications need to be able to scale. As the amount of visitors to a website is hardly predictable, it is a necessity that a SiteWorks engine, including SiteWorks applications, can be installed multiple times, to spread the load and create redundancy. This poses a problem, because when the SiteWorks engines all need to do the same, they need access to the same objects. Every object is loaded in the memory of the various SiteWorks engines, and therefore exists multiple times. The **synchronization module** solves this problem by interconnecting the SiteWorks engines. When an object is changed, the change is stored in the internal database. Then, the SiteWorks engine that changed the object give the other SiteWorks engines a signal that an object has been changed. The other SiteWorks engines then retrieve the new data of the object from the internal database. To make use of this functionality, it is a requirement that all SiteWorks engines use the same internal database. This is similar to other approaches (Koch, 1990 [3]).

2.3 Application Engineering

There are two ways for a developer to use SiteWorks as a development environment; the first option is to install a J2EE environment on the local machine and deploy the SiteWorks applications in it. SiteWorks needs a database, which needs to be installed locally or centralized. The advantage of this solution is that developers work in their own environment. Templates and other objects inside the system are automatically synchronized with files on the hard disk, which makes it possible to use configuration management tools, text editors and the browser interface next to each other. The second option is to set up a centralized installation. All developers use their browser to make changes in the system. Because everyone works on the same code base, this is

very fast way of working. However it can lead to unexpected and confusing errors, for when one developer introduces a bug, all other developers may run into it, or worse, will notice it immediately. The last scenario is best used when the development team is very small, and is not used very much at <GX>.

When the application built in SiteWorks is deployed, and changes need to be introduced, there is a three-way approach. Developers work locally on the source code base, making changes and committing the changes to the configuration management tool. When done, the changes are deployed on a test and acceptation server. Customers and end-users can test and accept the changes on this server. When they accept it, the changes are deployed on the live server, where they are tested once again. When changes in the SiteWorks system itself are made, the J2EE environment needs to be restarted. Applications inside SiteWorks can almost always be deployed without restarting, further reducing downtime of the live environment.

3 Implementing the WRDE

In section 1.2, we proposed that incorporating the WRDE concept in application servers would make developers much more productive. SiteWorks is a prototype of a system that uses these ideas. It's important to note that these design goals were a few of many design goals for SiteWorks. SiteWorks gives us a very good idea of the power of an application server using the WRDE concepts. The next paragraphs evaluate the WRDE concepts in SiteWorks.

3.1 Persistent Runtime

The main characteristic of the SiteWorks system is the built in transparent persistency. Persistency is not unique to SiteWorks. During the past few years numerous attempts are made to introduce persistence in the J2EE, .NET and other development and runtime environments (see reference for URL's with persistency tools for Java [4]). These attempts can be divided in two categories; the first is to create object-relational mappings and the second is to introduce transparent persistency.

Almost all persistence tools are based on object-relational mapping paradigm. When using those tools, a relational database and objects in the virtual machine are mapped to each other. When a change in an object is made, it is stored in the database, and the other way around, a change in the database implies a change in the corresponding objects. However, a problem is that object orientation and relational model are different paradigms, which do not fit at all times. Also, a lot of configuration or code generation is necessary to implement those mappings, which hamper development progress (Hou, 2002. A performance improvement by using caching is almost always a problem as a relational database can be changed without notice for the virtual machine. The objects inside the virtual machine need a signal to know they are stale, which isn't something a relational database provides, although research for it has been done (Machani, 1996 [5]).

The transparent persistency approach is not to think in terms of a relational database, but just to take care of the storage of the objects somewhere, not necessarily in a relational database. Prevayler ([6]) is an open source persistency handler that uses this

approach, without a database. Developers can define normal classes and create normal objects in Java, storage is handled by the Prevayler engine. When the Java Virtual Machine is restarted, the Prevayler engine reloads all the previous created objects. Because no connection with an external database and SQL manipulation is needed, the performance can be very fast. All objects need to be kept in memory though, which can create scalability problems for very large datasets.

SiteWorks is also using the second approach, but instead of building a persistent framework on top of the virtual machine, it implements the persistence inside the SiteWorks virtual machine.

This transparent persistency inside the SiteWorks engine itself is also known as orthogonal persistency, which cannot only be used in virtual machines, but also in operating systems (Dearle, 1994 [7]). A system like this is called a Persistent Object System or, when the system is more aimed at purely storing objects, a Persistent Object Store (Brown, 1988 [8]).

Attempts are made to do this in the Java Virtual Machine (Atkinson, 1996 [9]and 1998 [10]), but the Java Virtual Machine is not designed for such an environment. Java is a strongly typed compiled language. When changing a class, instances of this class need to change accordingly, but those instances are not available until the virtual machine is started. This causes a lot of problems to keep the object population compatible with the classes.

SiteWorks takes a different approach. Object type definitions can only be changed when the virtual machine itself is running. When an object type is changed, the objects belonging to this object type are changed accordingly and immediately. The developer will immediately see what the consequences are. Furthermore, as every change is atomic to the system, SiteWorks can change the instances belonging to the changed object type following strict defined rules, known by the developer. Because of the strict separation of the development and runtime environment for languages in Java and .NET, this is very hard to do in these languages. A developer can change all classes at once, which creates migration problems for the object population.

The main advantage of the transparent persistency in SiteWorks is the productivity gain for developers. They do not have to think about object persistence and can focus on the business problem, not on infrastructural problems like storage.

3.2 Development in a Runtime Environment

The second characteristic of the SiteWorks system is that development takes place inside the running Virtual Machine. Almost all popular languages have a strict separation between the development environment and the runtime environment. In SiteWorks, the developer can change templates, object types and objects while the engine is running. This approach makes it much easier to change object types, because the underlying engine can change instances of the object type immediately. Templates can be changed while a system is running, without the need to restart. By using the SiteWorks environment, the object population can be introspected and changed. This is especially useful when debugging, either in a test environment or on a deployed SiteWorks system.

A similar approach is tried with the Java Virtual Machine. Using classloaders and a special debug mode, it is possible to change classes ([11]) in a running Java Virtual Machine. When a class is replaced, problems occur when instances of the class al-

ready exist and the new class has new or changed field declarations. Also, non-literal field declarations that point to other classes can be a problem. Because a developer can change the whole class at once, the virtual machine will have to create a migration strategy for the 'old' objects. SiteWorks was designed from the ground up with these requirements in mind. As was said in paragraph 3.1, because the developer changes object types in the running environment one at the time, the changes are atomic, and the behavior of the system when making such a change can be strictly defined.

Advantages of this approach are the immediate visible effect of changes in templates, object types and objects, direct manipulation possible of the core entities and less restarting of the SiteWorks system necessary.

3.3 Web-Based Development

The third characteristic of SiteWorks is a fully web enabled user interface. This has two advantages. First, during development, a web browser is all that a developer needs, access to the development environment is very easy and can be done from almost any workstation with an Internet connection. This makes the development cycle a lot more efficient. When the system is deployed, developers can inspect and change the system using a web browser. Secondly, a clean separation between developers and system engineers is achieved. System engineers set up the system, engineers work on the application itself.

Admittedly, it does have some disadvantages too; because source code is centralized, configuration management tools can not be used as almost all of them are file based. Configuration and version management needs to be implemented in the system itself. Alternatively, developers may run a SiteWorks system locally. However, the advantage of separation of system engineering and development is then lost, the developer needs to know how to set up a SiteWorks system.

4 Usage of the WRDE

4.1 WebManager

The most prominent application built in SiteWorks is the content management and portal software WebManager. See figure 4 for screenshots.

In the left screenshot, the editor environment of WebManager is shown. Editors can manipulate content, layout and structure of the website, place functionalities on web pages etc. The left screenshot shows the resulting page. During the last 5 years about 100 WebManager installations are deployed, for large commercial companies like Planet Internet (largest Dutch ISP), Ajax, Mercedes and KPN (see appendix for the site addresses). It is also used by several Netherlands municipalities to implement their e-government initiative. WebManager itself consists of several components, like content management, interaction management, connectivity management and workflow (see figure 5). They are built in SiteWorks, but have several hooks to Java and other environments, to accommodate and leverage those platforms.

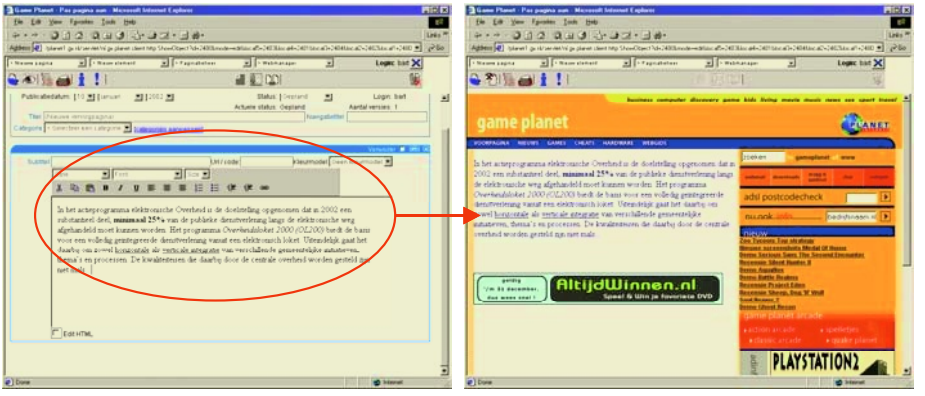

Fig. 4. A web application (right) engineered with WebManager (left)

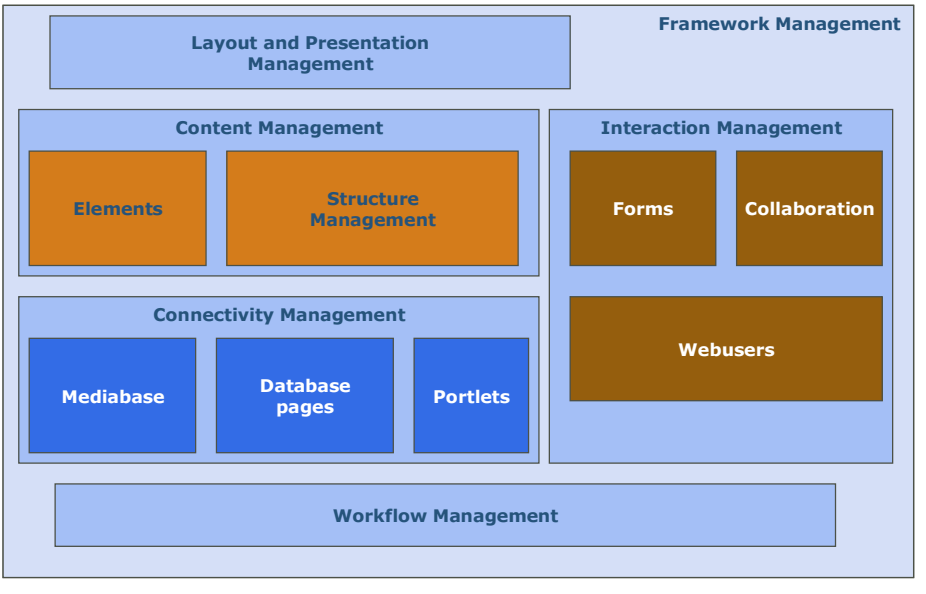

Fig. 5. The WebManager architecture

4.2 Support System & Systems Maintenance Manager

Using SiteWorks, a support system has been built for our support department at <GX>. With it, our support engineers can administrate problem reports, so called tickets, with accompanying knowledge and actions. Customers can log in on our extranet and can see the outstanding tickets, and the status they have. The system makes extensive use of the build in authorization module, and stores all data in SiteWorks objects. Because the developers do not need to care about the storage, we can adapt the system very fast, for example to give customers more options.

SiteWorks is also used for maintaining our installed base. This system keeps track of which version of WebManager runs where, what logins are, what customer logins and passwords are, how to get to the installed system. It also has connections with the support system, which is running in a separate SiteWorks installation. Because the system is web based, everyone can access the information very easily. This system also makes use of the build in authorization module, to prevent employees from viewing sensitive data.

5 Conclusions and Further Research

In this paper we stated that by letting developers using a WRDE, major productivity gains can be achieved. Evidence from various websites deployments supports this claim, especially with regard to the functionality and speed of development. The WRDE SiteWorks provides an important step forward in current web application engineering tools with transparent persistency handling, development in a runtime environment, and web based development tools. Customer reports and extremely high-availability rates of major sites in the Netherlands support the innovative directions in web technology.

We admit that we in the current stage we can only support the productivity gains in a qualitative manner. Therefore, further research is necessary to support this with more quantitative analysis, for example, by letting two development teams built the same functional application on different application server frameworks. Also, a serious challenge emerges when two populations of objects need to be merged (for example a development environment and a deployment environment). Object types and templates have a unique name and are merged accordingly. This is much harder for nameless objects.

We are currently working on several improvement of the SiteWorks system. For example, the development environment is now separate from the template system. Ideally, the development environment is built using those templates. The same can be done for the function library, as many functions inside the library can be accomplished using templates and special objects. These changes in the system could give a clearer and pure view of what exactly a WRDE is.

In the nearby future we hope to report on our current research program.

Acknowledgements

The authors wish to thank Arnoud Verdwaald and Mark Machielsen of <GX>, and their colleagues at Utrecht University, The Netherlands for their contributions and feedback on the SiteWorks system and on earlier versions of this paper.

References

1. An overview of the J2EE architecture: http://java.sun.com/j2ee/overview3.html
2. An overview of the .NET architecture: http://msdn.microsoft.com/netframework/technologyinfo/overview/default.aspx
3. Bett Koch, Tracy Schunke, Alan Dearle, Francis Vaughan, Chris Marlin, Ruth Fazakerley, Chris Barter (1990), Cache coherency and storage management in a persistent object system. Proceedings of the Fourth International Workshop on Persistent Object Systems, pages 99--109, Martha's Vineyard, MA (USA)
4. Overviews of Object Relational mapping and persistency tools available for Java. http://c2.com/cgi/wiki?ObjectRelationalToolComparison and http://www.cetus-links.org/oo_db_java.html
5. Salah-Eddine Machani (1996). Events in an Active Object-Relational Database System http://www.masi.uvsq.fr/rapports/1996/
6. Prevayler, a fast, transparent persistence, fault tolerant and load balancing architecture for Java objects. http://www.prevayler.org/
7. Alan Dearle, Rex di Bona, James Farrow, Frans Henskens, Anders Lindstrom, John Rosenberg, Francis Vaughan (1994). Grasshopper: An Orthogonally Persistent Operating System, Computing Systems volume 7, number 3, 289-312
8. A.L. Brown (1988). Persistent Object Stores, Ph.D thesis, University of St. Andrews, http://www-fide.dcs.stand. ac.uk/Info/Papers4.html#thesis
9. M.P. Atkinson, L. Daynès, M.J. Jordan, T. Printezis, S. Spence (dec 1996). An Orthogonally Persistent Java, ACM SIGMOD Record, 25, 4, pp68-75.
10. Malcolm Atkinson, Mick Jordan (1998), Providing Orthogonal Persistence for Java, Lecture Notes in Computer Science, volume 1445, page 383
11. Hotswap options in the Java Platform Debugger Architecture: http://java.sun.com/j2se/1.4.1/docs/guide/jpda/enhancements.html#hotswap
12. Daqing Hou, H. James Hoover, Eleni Stroulia (may 2002). Supporting the Deployment of Object-Oriented Frameworks, Proceedings of the 14th International Conference CaiSE 2002, LNCS 2348, Springer Verlag, page 151-166
13. Mike P. Papazoglou (2003), Web Services and Business Transactions. World Wide Web vol. 6, nr. 1, pp. 49-91.
14. P. Fraterneli (1999), Tools and approaches for developing data-intensive web applications: a survey, ACM Computing Surveys, vol. 31, nr. 3, pp. 227 – 263.
15. Stefano Ceri, Piero Fraternali, Aldo Bongio, Marco Brambilla, Sara Comai, Maristella Matera (2002), Designing Data-Intensive Web Applications. Morgan Kaufmann
16. L. Liu, E. Yu, Designing Web-Based Systems in Social Context: A Goal and Scenario Based Approach, 14th International Conference on Advanced Information Systems Engineering (CAiSE'02), Toronto, May 27-31, 2002. LNCS 2348 Springer Verlag. pp. 37-51.
17. F.J. Garcia, A.B. Gil, N. Moreno, B. Curto (2002), A web-based e-commerce facilitator intermediary for small and medium enterprises: a b2b/b2c hybrid proposal. In: Proceedings of the 3rd Int. Conf. EC-Web 2002. LNCS 2455, Springer Verlag.

Appendix – Websites Running on the SiteWorks System

- Ajax – A consumer oriented fan site of the famous Dutch football club: http://www.ajax.nl/
- KPN Breedbandportal – A consumer broadband site commercialized by KPN: http://breedbandportal.kpn.com/

- Planet Technologies – The web portal of the largest ISP in the Netherlands: http://www.planet.nl/
- Daimler Chrysler Nederland – Consumer oriented website for their main brand Mercedes: http://www.mercedes-benz.nl/
- Gemeente Maastricht – Municipality Maastricht website: http://www.maastricht.nl/
- Gemeenten Vlissingen – Municipalities Vlissingen website: http://www.vlissingen.nl/
- Unicef Nederland – Consumer site for Unicef the Netherlands: http://www.unicef.nl/