

# BIOINFORMATICS

Electronic edition <http://www.bioinformatics.oupjournals.org>

VOLUME 17  
NUMBER 1  
JANUARY 2001  
PAGES 83–94

## ***ISYS: a decentralized, component-based approach to the integration of heterogeneous bioinformatics resources***

*A. Siepel\**, *A. Farmer*, *A. Tolopko*, *M. Zhuang*, *P. Mendes*, *W. Beavis* and *B. Sobral*

*National Center for Genome Resources, 2935 Rodeo Park Drive East,  
Santa Fe, NM 87505, USA*

Received on May 15, 2000; revised on August 11, 2000; accepted on August 14, 2000.

\*To whom correspondence should be addressed.



GO BACK

CLOSE FILE

# Abstract

**Motivation:** *Heterogeneity of databases and software resources continues to hamper the integration of biological information. Top-down solutions are not feasible for the full-scale problem of integration across biological species and data types. Bottom-up solutions so far have not integrated, in a maximally flexible way, dynamic and interactive graphical-user-interface components with data repositories and analysis tools.*

**Results:** *We present a component-based approach that relies on a generalized platform for component integration. The platform enables independently-developed components to synchronize their behavior and exchange services, without direct knowledge of one another. An interface-based data model allows the exchange of information with minimal component interdependency. From these interactions an integrated system results, which we call ISYS<sup>TM</sup>. By allowing services to be discovered dynamically based on selected objects, ISYS encourages a kind of exploratory navigation that we believe to be well-suited for applications in genomic research.*

**Availability:** *A 'developer's kit' for creating software that uses the ISYS platform is available at [www.ncgr.org/research/isys/devrel.html](http://www.ncgr.org/research/isys/devrel.html). It includes a more refined and fully documented version of the platform and a starting set of ISYS-ready components. Up-to-date descriptions of the ISYS project can be found at [www.ncgr.org/research/isys](http://www.ncgr.org/research/isys).*

**Contact:** [isys@ncgr.org](mailto:isys@ncgr.org)

Abstract

Introduction

Illustration of ISYS

System architecture

Discussion

Acknowledgements

References



GO BACK

CLOSE FILE

# Introduction

A central problem of bioinformatics is to synthesize new scientific knowledge about genome structure, gene function, and evolutionary relationships of organisms from massive amounts of low-level data. Too often, however, synthesis is impeded by incompatibilities among heterogeneous databases and computer programs, which make it cumbersome to find critical connections among data from different sources (Ritter, 1994; Markowitz and Ritter, 1995; Davidson *et al.*, 1995; Baker *et al.*, 1998).

Several bioinformatics groups have attacked the problem of heterogeneity on various fronts. Investigators have developed multi-database query systems that span heterogeneous repositories (Baker *et al.*, 1998; Chung and Wong, 1999; Davidson *et al.*, 1999; Markowitz *et al.*, 1996, 1999), data warehouses that homogenize data (Ritter, 1994; Aberer and Hemm, 1996; Leser *et al.*, 1998) and that do not homogenize data (Etzold *et al.*, 1996; Carter *et al.*, 1999), standards for data representation and exchange (Life Sciences Research Task Force, 1997; Slidel, 1998; Benton, 2000; Bio-Ontology, 2000), pilot projects employing such standards (Rodriguez-Tomé and Lijnzaad, 1999; Jungfer *et al.*, 1999), and systems that interlink web-based resources (Fujibuchi *et al.*, 1997; Sirotkin, 1999). One way to categorize these efforts is to divide them into top-down and bottom-up approaches. Top-down approaches, such as data warehousing, are characterized by central control, enforced homogeneity, and coordinated development. They offer the promise of tight integration and proper resolution of the issues of data quality and consistency that plague other efforts

Abstract

Introduction

Illustration of ISYS

System architecture

Discussion

Acknowledgements

References



GO BACK

CLOSE FILE

(Leser *et al.*, 1998), but they pay a high price in terms of system flexibility and development time. Because of the lack of unity among independently-funded bioinformatics groups, large-scale top-down solutions (that is, ones not limited to relatively small data sets, such as those for specific organisms) probably will not be feasible until global standards for data representation are in place and widely observed; and even if the pursuit of stable standards is not futile (Markowitz and Ritter, 1995; Davidson *et al.*, 1995), they will take many years to establish. Meanwhile, there are pressing needs in the biological research community for systems to enable information integration.

Because we are interested in large-scale integration, spanning both species and biological ‘data types’ (e.g. sequences, maps, pathways), we have looked toward bottom-up solutions, which ‘embrace diversity’ rather than denying it (Benton, 2000). Even among these solutions, one sees various levels of homogenization and centralization. At one extreme, the approach of hyperlinking separately-developed web pages is exceedingly forgiving of heterogeneity and highly decentralized, but is limited in terms of user interface capability (Helt *et al.*, 1998; Fischer *et al.*, 1999) and query power (Karp, 1996; Baker *et al.*, 1998). At the other extreme, multi-database query approaches, which generally require a central ‘brain’ to coordinate and optimize queries, allow for powerful declarative queries but either require imperfect just-in-time mappings to definitive ontologies (Karp, 1996; Baker *et al.*, 1998, 1999), or force users to learn schematic details of source databases (Markowitz *et al.*, 1996; Chung and Wong, 1999; Baker *et al.*, 1998). These systems also can be

Abstract

Introduction

Illustration of ISYS

System architecture

Discussion

Acknowledgements

References



GO BACK

CLOSE FILE

ineffective when operating with unstandardized source databases, which require human intervention for accurate mappings (Leser *et al.*, 1998).

A middle road is to define standard interfaces by which heterogeneous components (which may include databases as well as other software tools) can interoperate, but to allow for variability in implementation, and complete freedom in the assembly of systems from components (Life Sciences Research Task Force, 1997; Slidel, 1998; Benton, 2000). This approach does not address problems of object identification and interdatabase consistency (Leser *et al.*, 1998) and does not directly enable complex declarative queries (although it is not incompatible with them). However, it does allow heterogeneous resources to exchange data with minimum homogenization, via the principle of component encapsulation (Szyperski, 1998), and it is conducive to decentralization. In addition, component-based approaches offer many important development and maintenance benefits (Searls, 1995; Goodman *et al.*, 1995; Boyle, 1998; Szyperski, 1998), and if properly standardized, could revolutionize software development in bioinformatics by enabling developers to build systems from components purchased in an ‘open market’ (Slidel, 1998). Finally, component-based development aligns well with user-interface development (Searls, 1995; Boyle, 1998; Fischer *et al.*, 1999), and thus facilitates the development of visually intuitive and interactive systems. Note that these solutions can be more or less ‘top-down’ depending on the level of abstraction of the inter-component interfaces. There is tension between greater abstraction, which can result in loss of information during exchanges between components,

Abstract

Introduction

Illustration of ISYS

System architecture

Discussion

Acknowledgements

References



GO BACK

CLOSE FILE

but can increase the flexibility of systems and the degree of freedom enjoyed by implementers of components; and greater specificity, which improves information exchange but decreases flexibility (in the extreme, highly specific interfaces reduce, in terms of data exchange, to a canonical schema).

Many current component-based integration solutions for bioinformatics (Life Sciences Research Task Force, 1997; Slidel, 1998; Rodriguez-Tomé and Lijnzaad, 1999; Jungfer *et al.*, 1999; Benton, 2000) are based on CORBA (Object Management Group, 1996), a powerful technology for platform- and language-independent inter-process communication. CORBA, however, is primarily a server-side technology, being overly ponderous and complex for most intra-process, client-side integration needs (there are also problems relating to the redeployment of commercial ORBs). Higher level component models such as Enterprise JavaBeans™ (EJBs) are similarly better suited for server components and inter-process communication than for client-side components (Monson-Haefel, 1999). There do exist component models more suitable for components running in the same process space, such as JavaBeans™. By themselves, JavaBeans are essentially Java™ classes adhering to interfaces that allow them to be inspected by and manipulated within component assembly tools. With the aid of Sun Microsystems Inc.'s InfoBus (InfoBus, 2000), however, beans can exchange data dynamically with minimal interdependencies. JavaBeans connected using the InfoBus are not sufficient by themselves as a platform for client-side integration of bioinformatics components, in our view, because they lack the ability to broker services

Abstract

Introduction

Illustration of ISYS

System architecture

Discussion

Acknowledgements

References



GO BACK

CLOSE FILE

(see below), they are data-centric (i.e. they are not designed for use in visual synchronization, although they could be used that way), and they do not provide the infrastructure for communicating complex biological data among components (also see below).

Our approach to information integration relies on a newly-created platform that allows separately-developed client-side components to interoperate without direct knowledge of one another. Autonomous components exchange events and services, according to a small set of simple rules, and in doing so cause the emergence of an 'integrated system' (called ISYS™) with no centralized control or fixed set of properties. This system, consequently, is extremely flexible, configurable, and extensible. The platform resides on the client side, but local or remote servers of any kind are accessible through server proxies, which cause them to appear like any other client-side resource. A flexible data model allows components to exchange biological information without being tied to a single consistent mode of representation. We have developed a prototype demonstrating this approach, which consists of the integration platform, a starting set of client-side components, and various server-side database and computational resources. As will be seen below, the prototype supports a kind of exploratory navigation that we believe to be well-suited for the needs of genomic researchers.

**Abstract**

**Introduction**

**Illustration of ISYS**

**System architecture**

**Discussion**

**Acknowledgements**

**References**



**GO BACK**

**CLOSE FILE**

# Illustration of ISYS

We illustrate the current prototype by describing how it might be used. We have selected a scenario that begins with a genomic map for *Arabidopsis thaliana*, proceeds to an annotated sequence, then by way of sequence similarity, to an associated metabolic pathway. This scenario is representative of the kind of comparative and functional analysis we believe an integrated bioinformatics system should support. It provides a good example of what is possible with the ISYS platform when it is used to integrate existing database and analysis resources with relatively simple software components.

The user begins with a Quantitative Trait Locus (QTL) on chromosome 1 of *Arabidopsis*, hoping to identify specific genes that may be functionally related to the trait in question. She enters the system by selecting the MapViewer from a panel of buttons that represent entry points to the system. After choosing to view chromosome 1 of *Arabidopsis*, she is presented with a browser displaying a genetic map and a physical map in parallel (Fig. 1). She zooms in on a section of the map proximate to the QTL and selects a bar in that region representing a sequenced clone. Next, she chooses to display sequence annotations for the clone, as represented in a sequence database. The system displays them at proper scale, within the MapViewer (Fig. 1).

The user scans the annotations and selects a coding region for further examination. The coding region coincides with the QTL, but lacks functional annotation. With a click of the right mouse button, a popup menu appears and displays a list of registered services capable of operating on the selected coding

Abstract

Introduction

Illustration of ISYS

System architecture

Discussion

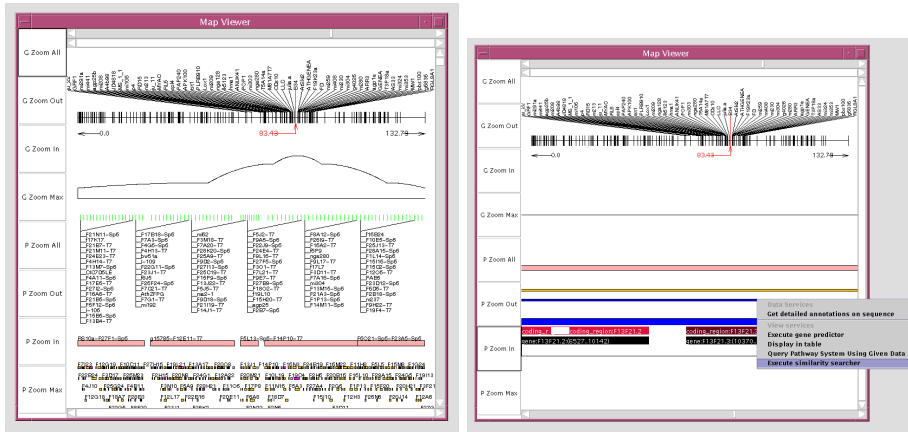
Acknowledgements

References



GO BACK

CLOSE FILE



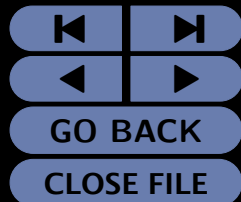
**Fig. 1.** Screen shots of MapViewer, when initially invoked on chromosome 1 of *Arabidopsis thaliana* (top), and when zoomed in on a particular clone (bottom). The MapViewer shares code with a web-based tool for The *Arabidopsis* Information Resource (TAIR, 1999). It allows the user to zoom and scroll a genetic map and a physical map, each independently of the other. Markers, probes, contigs, and clones are shown, as well as a generic graphing component (here plotting only a random function, but intended eventually to display information such as GC percentage or density of identified genes). Data is drawn from TAIR transparently, via a server proxy. The curve in the middle of the panel represents a hypothetical QTL. At right, the SequenceViewer is shown embedded within the MapViewer, displaying annotations for a particular clone. Its behavior is coordinated with the MapViewer in terms of scroll position and zoom level. Annotations (shown as colored bars) are retrieved from the Genome Sequence DataBase (Skupski *et al.*, 1999) via another server proxy (any equivalent source of data could be substituted). The user has just right-clicked on one of the sequence features, to perform Dynamic Discovery. The services shown have been identified at run-time according to the set of components present and responding to the user's request. Some services refer to sequences rather than sequence features, because features are interpreted as equivalent to corresponding subsequences.

Abstract  
Introduction

Illustration of ISYS  
System architecture

Discussion  
Acknowledgements

References



region (Fig. 1), representing various possible ways to proceed. This mechanism, which we call *Dynamic Discovery*, is central to ISYS; it enables a user to find various paths through the system dynamically, based on selected data and what components are present. Here, the user chooses to perform a similarity search, hoping that apparent homologs to the gene of interest may provide clues about its function (note that, had the clone lacked annotated genes, she could first have estimated them by choosing to invoke a registered gene prediction service; GenScan (Burge and Karlin (1997) is available in the prototype). She is presented with a separate component, the SimilaritySearcher (Fig. 2), which allows her to launch a search, and to browse its results. This component is ‘seeded’ automatically with the sequence segment corresponding to the selected coding region. When a browser showing search results appears, the best hits to the query sequence also appear as features in the MapViewer; moreover, the selection and visibility of corresponding objects are synchronized in the two components. Synchronization occurs by event exchange, without components having direct knowledge of one another.

More than two components can be synchronized, as can be seen when the user selects the hits in the SimilaritySearcher’s browser and invokes by Dynamic Discovery an option to show taxonomic relationships (Fig. 2). In this case, three components representing three different perspectives on the same data—the MapViewer a structural one, the TaxonomyViewer an evolutionary one, and the SimilaritySearcher a comparative one—behave in concert, and the user is provided with a display richer than the sum of its parts. For example,

Abstract

Introduction

Illustration of ISYS

System architecture

Discussion

Acknowledgements

References



GO BACK

CLOSE FILE

selecting a higher-level taxonomic node in the TaxonomyViewer (e.g. the family *Brassicaceae*) causes all subsumed species to be selected; and by their synchronous behavior, the SimilaritySearcher and SequenceViewer reflect a taxonomic grouping, despite having no knowledge of hierarchical relationships among species. As a result, a user can explore how taxonomic grouping, degree of similarity, and structural location relate to one another. The user could continue further in the vein of sequence analysis and evolutionary relationships, as described in Fig. 3.

To address the question of functional significance, the user retrieves sequence annotations for apparent homologs to the original query sequence, and then has enough information to navigate to PathDB (Waugh *et al.*, 2000). There she can see the role of her gene in the context of cellular metabolism (Fig. 4). Variations on the same theme could be repeated for other coding regions aligned with the original QTL.

In this example, the system allows the user to begin with a bird's eye view of genome structure, to plunge into the details of sequences and sequence annotation, and with help from comparative genomics, to emerge with a broad functional perspective on a structural entity. What is perhaps most significant, however, is that this and many other sweeping paths across species, data types, and biological perspectives are made possible by allowing a few simple types of integration among independent, decoupled, and unexceptional components.

**Abstract**

**Introduction**

**Illustration of ISYS**

**System architecture**

**Discussion**

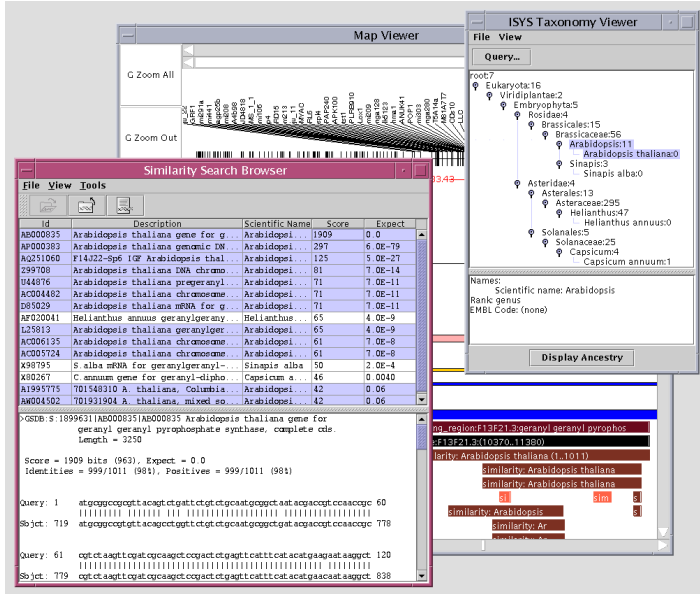
**Acknowledgements**

**References**



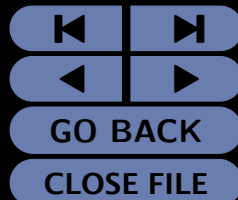
**GO BACK**

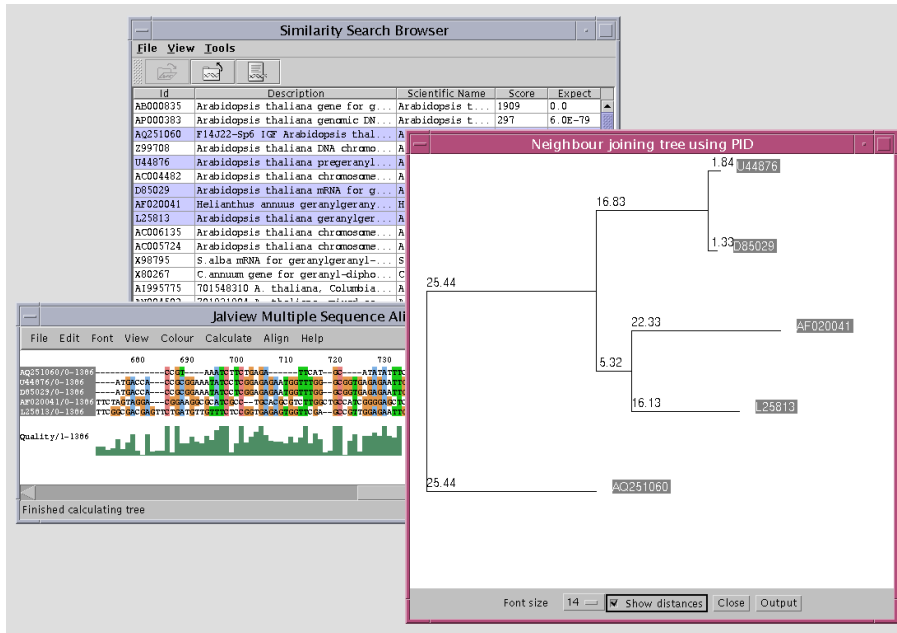
**CLOSE FILE**



**Fig. 2.** SimilaritySearcher and TaxonomyViewer, invoked by Dynamic Discovery from MapViewer. The user has launched a BLAST search (Altschul *et al.*, 1990) using a search launcher (not shown); the search has been executed transparently by a server proxy that calls the National Center for Genome Resources' CORBA-based Distributed Analysis system (Inman, 2000); then the browser shown here has appeared. The browser's top panel displays the best hits to the search, along with corresponding species as annotated in GSDb (retrieved via another service, unknown to the user), and the bottom one shows the local alignment describing the first selected item. The existence of the search results has been broadcast as an event, and picked up by the SequenceViewer, which shows them as sequence annotations (see bottom right). From the browser, the user has invoked the TaxonomyViewer (top right) to describe the hierarchical relationships among species. Taxonomic data are retrieved transparently from a relational version of the National Center for Biotechnology Information's taxonomy database (NCBI, 2000), via a server proxy. As nodes are selected in the TaxonomyViewer, synchronous selection of corresponding search hits occurs in the SimilaritySearcher. As subtrees are collapsed in the TaxonomyViewer (like in a file browser), corresponding hits are 'filtered out' and become invisible in the SimilaritySearcher. Selection and visibility events are communicated transitively to the SequenceViewer for three-way synchronization. Note that synchronization and service invocation occur despite components having no direct knowledge of one another.

Abstract  
 Introduction  
 Illustration of ISYS  
 System architecture  
 Discussion  
 Acknowledgements  
 References





**Fig. 3.** JalView multiple sequence alignment editor (JalView, 1998), showing global alignment of a subset of apparent homologs from Fig. 2. The user has invoked an option to ‘align sequences’ from the SimilaritySearcher, which spawned a simple interface for CLUSTAL W (Thompson *et al.*, 1994); she has defined and launched the search, which has been executed by a server proxy and local server; and JalView has been spawned. JalView was ‘wrapped’ for integration into ISYS, with the wrapper handling registry with the platform, and broadcast and reception of events. Here the user has created a neighbor-joining tree of the aligned sequences (using JalView), which is coordinated in terms of selection with the multiple sequence alignment and all of the other ISYS components.

Abstract

Introduction

Illustration of ISYS

System architecture

Discussion

Acknowledgements

References



GO BACK

CLOSE FILE

# System architecture

Fig. 5 is a schematic diagram showing a portion of the system architecture. It shows several components on the client side, integrated by way of the 'ISYS Platform'. Here we consider in turn the platform, the exchange of services, the exchange of events, how non-platform components fit into the system, and the data model that unifies the system.

## *The platform*

The 'ISYS Platform' consists of two components that are responsible for the integration and management of all others: the *ClientBus* and the *ISYS Client Environment (ICE)*. The ICE is the simple user interface that allows a user to invoke components directly; we will not describe it further here. The ClientBus is responsible for all communication between components, and is the heart of the system. It is actually a combination of an Event Channel (sometimes called a bus) and a Broker (Buschmann *et al.*, 1996). As an Event Channel, the ClientBus enables components to register as 'listeners' for particular types of events, or occurrences, and react to them in prescribed ways. As a Broker, it allows components to request and provide services to one another, as defined by service name, or by types and attributes of objects for which services are requested. In both cases, components interoperate without having direct knowledge of one another, to promote flexibility.

The interface to the ClientBus supports five essential capabilities: (1) registry

Abstract

Introduction

Illustration of ISYS

System architecture

Discussion

Acknowledgements

References



GO BACK

CLOSE FILE

of a component as a listener for a particular class of events; (2) broadcast of an event, generally linked to a specific underlying data object; (3) registry of a component as a provider of particular services; (4) request of services by name; and (5) request of services applicable for a particular object. Of these, the first two relate to events, and the final three to services.

### *Exchange of events*

Event exchange has been used widely as a mechanism to allow synchronized behavior among components (Szyperski, 1998). The distinguishing characteristic of the Event Channel, a variant of the standard Observer pattern (Gamma *et al.*, 1995), is that it allows components to exchange events without registering directly with one another. Instead they remain decoupled and interact only with the Event Channel, which behaves as a mediator among components (Buschmann *et al.*, 1996).

In ISYS any component may register with the bus as a listener for any type of event (the prototype contains only three such types: *ItemUpdatedEvent*, *ItemSelectionEvent*, and *ItemVisibilityEvent*). Event objects generally are given references to associated data objects. Once a component is registered, all events of the specified type will result in calls to the component's listener method, as long as the two components are defined as being synchronized (the system allows deactivation of synchronization where it would be unnecessary or confusing).

Abstract

Introduction

Illustration of ISYS

System architecture

Discussion

Acknowledgements

References



GO BACK

CLOSE FILE

For example, in the scenario described above (Fig. 2), when the user selected a matching sequence in the SimilaritySearcher, that component generated an ItemSelectionEvent and ‘fired’ it by calling a method on the ClientBus. This event held a reference to a data object representing the selected homolog. The ClientBus, in turn, recognized that the SequenceViewer was synchronized with the SimilaritySearcher (as requested by the SequenceViewer when it originally spawned the SimilaritySearcher), and called the SequenceViewer’s registered listener for ItemSelectionEvents. Finally, this listener, within the SequenceViewer, inspected the data object for an identifier, located the corresponding object among the features it was displaying, and highlighted it as selected.

### *Exchange of services*

The Broker pattern (Buschmann *et al.*, 1996) is similar to the Event Channel in that it allows decoupled components to interact, but different in that it is a ‘pull’ mechanism rather than a ‘push’ one—i.e. the receiving component is provided services when it requests them, rather than reacting to circumstances beyond its control. The Broker pattern is best known for its central role in CORBA (Object Management Group, 1996) and generally is applied to systems having distributed components.

As used in ISYS, the pattern differs from its classic form (Buschmann *et al.*, 1996) in three main ways. First, services are objects, passed via the broker from service providers to service consumers. They encapsulate their behavior using

Abstract

Introduction

Illustration of ISYS

System architecture

Discussion

Acknowledgements

References



GO BACK

CLOSE FILE

the Command pattern ([Gamma et al., 1995](#)). This is a relatively minor change that allows services to be passed among components, and that causes specific services (analogous to methods), rather than servers with specific interfaces (collections of related methods), to be the unit of exchange. Second, the Broker runs on the client, and allows direct interoperation only among client-side components. As a result, client-side and server-side proxies handling tasks associated with network communication (as described in [Buschmann et al., 1996](#)), are not necessary. Third, in addition to exchange of services by name, the ClientBus supports Dynamic Discovery, by which registered service providers evaluate a particular data object and respond whether or not their services are suitable for it, and suitable services are forwarded to a requester.

To understand the mechanics of Dynamic Discovery, consider the case described in [Fig. 1](#). The SequenceViewer has identified a data object associated with the selected graphical object. It has requested from the ClientBus all services capable of operating on this object. The ClientBus has forwarded the request to all registered service providers. Each service provider has inspected the object, and if it finds its type recognizable and sufficient data present, has created appropriate Service objects and passed them to the ClientBus. The ClientBus has returned these service objects to the SequenceViewer. The SequenceViewer has displayed their descriptions in a pop-up menu. The user has selected one of these descriptions, and finally, the SequenceViewer has invoked the corresponding service, passing it the original data object.

**Abstract**

**Introduction**

**Illustration of ISYS**

**System architecture**

**Discussion**

**Acknowledgements**

**References**



**GO BACK**

**CLOSE FILE**

## How components fit into the system

Non-platform components directly belonging to the system (i.e. components that are neither part of the platform nor external servers) must implement at least one of two interfaces: *Client* and *ServiceProvider*. Implementers of *Client* may fire and receive events, and implementers of *ServiceProvider* may register and provide services. Any component may request a service. Also, a single component may implement both interfaces (although this is not usually done).

*ServiceProviders* usually act as server proxies or client factories. Server proxies are gateways to servers. They use the Proxy pattern ([Buschmann et al., 1996](#)) to provide a layer of insulation between the system and external resources. This is valuable in two ways: if the interface to an external resources changes, alterations to the system may be confined to the server proxy; and server proxies that fulfill the same abstract responsibilities (e.g. a server proxy that executes BLAST searches at NCGR, and one that executes BLAST searches at NCBI) can be interchanged without perturbing the rest of the system. Client factories are responsible for generating instances of clients.

Servers are necessary for the system but are external to it. Any resource that can be called by a server proxy, generally as a provider of data, can function as a server. Such a resource may exist on the Internet, on a local network, or on the same machine as the client. (In general, the problem of scalability can be addressed this way: processes involving large quantities of data or intensive communication are handled on appropriate servers, which are integrated in the client-side environment via server proxies). Communication

Abstract

Introduction

Illustration of ISYS

System architecture

Discussion

Acknowledgements

References



GO BACK

CLOSE FILE

with remote servers can occur using any common protocol (in the prototype HTTP, CORBA, and Java RMI™ are used). Separately-developed software tools generally are added to the system as clients or servers. User interface components are added by wrapping them with a simple, lightweight Java class, that implements the appropriate interfaces and delegates calls to the legacy tool. If the tool is to invoke services or fire events, its user interface events must be intercepted in some way. This is trivial to accomplish if the component is written in Java and its source code is available, or if a sufficient Applications Programming Interface (API) is provided. If not, intercepting events is still possible although more difficult. An accompanying client factory, also small and simple, must be created to support the client. Data providers, either in the form of databases or analysis tools, are generally added as servers, and accessed via server proxies. Since server proxies have complete freedom in the retrieval of information, they may fulfill their responsibilities without using a true server (in the conventional sense of an interacting local or remote process). For example, to retrieve information a server proxy may invoke methods of a utility library, or spawn a child process.

### *The data model*

As we have seen, ISYS depends in several ways on having a global view of the definitions of and relationships among biologically significant entities. Such a global data model is essential for Dynamic Discovery, for the recognition of the subjects of events, and to define objects passed to services. We have

**Abstract**

**Introduction**

**Illustration of ISYS**

**System architecture**

**Discussion**

**Acknowledgements**

**References**



**GO BACK**

**CLOSE FILE**

avoided defining a highly-detailed, comprehensive schema, in the belief that it would compromise the flexibility of the system. Instead we settled on a solution intended to maximize the insulation of each evolving component from changes to the other. Our solution uses Java interfaces to define in highly general and abstract terms the essential entities to be exchanged among components, while allowing components complete freedom in the implementation of these interfaces.

Our data model is based on the idea that there exist a relatively small number of fundamentally important classes of biological objects that tend to resist major schematic variation: e.g. DNA sequences, genes, proteins, enzymes, pathways, maps, mappable elements, taxonomic designations. Along with their core attributes, these classes form a common denominator for biological databases, with differences among schemas tending to occur in less essential details (e.g. how feature locations are represented, what kinetic parameters are represented for enzymes). Because these fundamental classes tend to reflect stable and established models and concepts, relationships between components generally depend on them, not on the lesser details (their identities become evident when one analyzes the needs of each component from the others). For example, the entities in GSDB ([Skupski \*et al.\*, 1999](#)) and PathDB ([Waugh \*et al.\*, 2000](#)) are associated primarily according to relationships between genes and enzymes, and those in GSDB and [TAIR \(1999\)](#) primarily according to relationships between maps and sequences.

Our design approach is to expose only the fundamental classes in component

**Abstract**

**Introduction**

**Illustration of ISYS**

**System architecture**

**Discussion**

**Acknowledgements**

**References**



**GO BACK**

**CLOSE FILE**

interfaces, and to encapsulate manipulation of the less important details within the components. The cost of this is small, compared to the advantages it offers in system flexibility, because we find that the details tend not be important across component boundaries. For example, we encapsulate a search for enzymes having particular kinetic properties within a pathway-searching component. Next, we look across components to transform the resulting set of enzymes into a set of associated genes in GSDB, using the relationship between the fundamental classes for enzymes and genes. The kinetic parameters are generally important only indirectly for cross-database uses, so we do not expose them in the pathway component's interface, to avoid unnecessary interdependencies with other components. In addition, if the user is operating in an exploratory mode, trying to learn from an unfamiliar data set, it makes sense to let him see the intermediate set of enzymes anyway, since he may choose to filter them (or add to them) in some way before proceeding to the associated genes.

We have used Java interfaces to implement this approach. An interface can be defined for each of the fundamental classes, representing its attributes (via accessor methods) and essential behaviors. Interfaces can inherit multiply from one another, classes can implement multiple interfaces, and interfaces can include methods that declare associations with other interfaces. Consequently, a single object can simultaneously represent multiple fundamental classes. For example, a class called *Homolog*, used by the SimilaritySearcher, implements both the fundamental interface *IsysSequence* and an interface *HasTaxon* that

**Abstract**

**Introduction**

**Illustration of ISYS**

**System architecture**

**Discussion**

**Acknowledgements**

**References**



**GO BACK**

**CLOSE FILE**

declares a close association with an object having the fundamental interface *IsysTaxon*. When the user performs Dynamic Discovery on an object of class *Homolog* he will find services responding in terms of both interfaces, and thus will be able to manipulate his object as a sequence or as a taxon. Similarly, when an event is fired that refers to such an object, other components will react to both the corresponding *IsysSequences* and *IsysTaxons* (Fig. 6).

Note that this way of doing things differs from the standard Model-View-Controller (MVC) approach (Buschmann *et al.*, 1996). Instead of observing a shared model, components exchange information about private models and views in the *lingua franca* of the Java interfaces (components can still use the MVC pattern internally). Since they can implement these interfaces however they choose, their private models can be kept at a high-level or made arbitrarily rich. For example, the *TaxonomyBrowser* has a much richer implementation of the interface *IsysTaxon* than does the *SimilaritySearcher*; but because certain objects in both components are understood to be of type *IsysTaxon*, they can be exchanged readily. Furthermore, the *TaxonomyBrowser* can change its implementation however it wants without disrupting the *SimilaritySearcher*, as long as the interface is kept the same.

It is also worth noting that this data model does not represent a competing standard for representing bioinformatics data, and is compatible with developing standards (Life Sciences Research Task Force, 1997). The *ISYS* data model is generally more abstract and less comprehensive than other models and can be easily mapped to subsets of them. Components may employ more

Abstract

Introduction

Illustration of ISYS

System architecture

Discussion

Acknowledgements

References



GO BACK

CLOSE FILE

detailed standards for intra-component use, but communicate with other ISYS components using the ISYS data model. They may even choose to interact with one another directly at lower levels at the cost of decreased insulation from changes to one another.

We have encountered some problems with this approach, and are currently revising the data model. For example, the idea of universal fundamental classes is an over-simplification: some classes are more essential than others, but different components have different ‘views’ of what their core attributes are. The next version of our data model will associate attributes with fundamental classes at run-time (by aggregation), rather than coding them into interfaces. Consequently, the set of attributes belonging to objects of a given type will not be fixed, and components will not need to be recompiled when the data model is extended (the revised model is described in [Siepel \*et al.\*, 2000](#)).

**Abstract**

**Introduction**

**Illustration of ISYS**

**System architecture**

**Discussion**

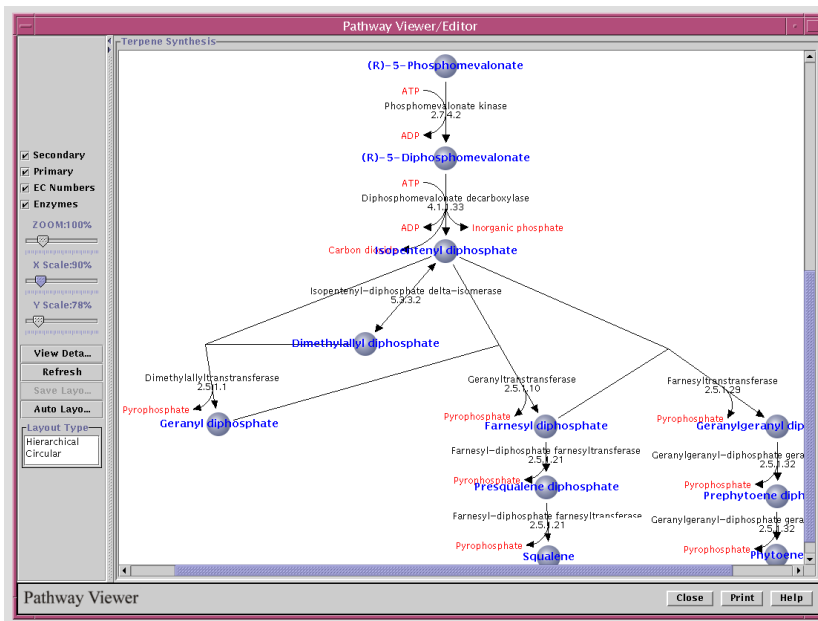
**Acknowledgements**

**References**



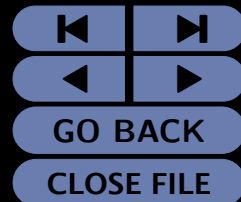
**GO BACK**

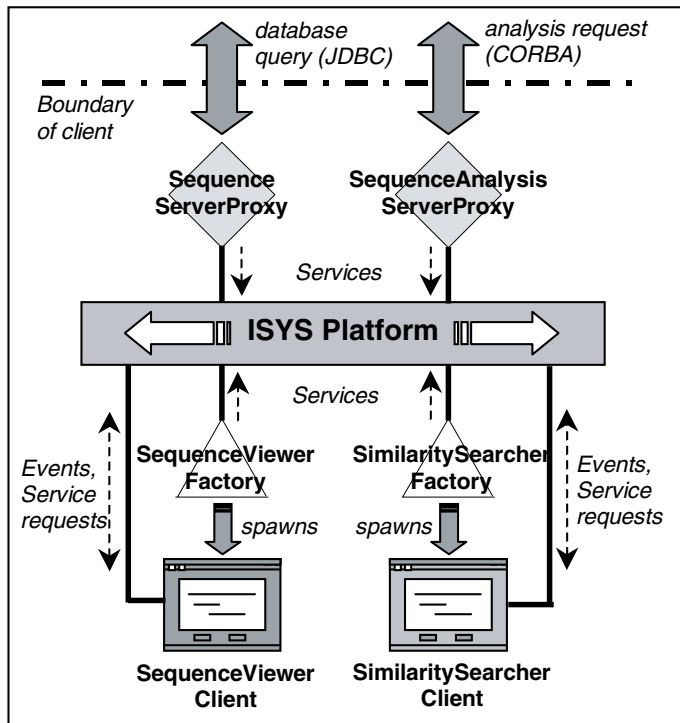
**CLOSE FILE**



**Fig. 4.** PathwayViewer showing a diagram of metabolic pathway related to query sequence from Fig. 2. The PathwayViewer is slightly adapted from the standard version (Waugh *et al.*, 2000). It has been spawned by a service that uses a server proxy to find objects in PathDB associated with a sequence or set of sequences, based on sequence annotations that include EC numbers. Although the query sequence did not have such annotation, some of its apparent homologs did.

Abstract  
 Introduction  
 Illustration of ISYS  
 System architecture  
 Discussion  
 Acknowledgements  
 References





**Fig. 5.** Schematic of system architecture, showing some of the components involved in the scenario described in Fig. 2 and their interactions with the ISYS Platform. Two Clients and four Service Providers (two ServerProxies and two ClientFactories) are shown. Server-side resources are not shown, but interactions with them are indicated at the top.

Abstract

Introduction

Illustration of ISYS

System architecture

Discussion

Acknowledgements

References



GO BACK

CLOSE FILE

```

/** Interface for "fundamental class" of DNA sequences. */
public interface IsysSequence {
    public String getICAccession ();
    public String getSequenceText ();
    public String getDescription ();
}

/** Interface for "fundamental class" of taxonomic nodes. */
public interface IsysTaxon {
    public String getRank (); // e.g., genus, species
    public String getName (); // e.g., Arabidopsis, thaliana
}

/** Interface showing association with IsysTaxon. */
public interface HasTaxon { public IsysTaxon getTaxon (); }

/** Simplified version of Homolog class used by SimilaritySearcher,
 * which represents an apparent homolog to a "query sequence" based
 * on the results of a similarity search.
 */
public class Homolog implements IsysSequence, HasTaxon {
    IsysSequence querySequence, backgroundSequence;
    Vector pairwiseAlignments; // set of local alignments found by similarity
                               // searching algorithm (e.g., HSPs in BLAST)

    // ...

    // methods specific to Homolog
    public getQuerySequence () { return querySequence; }
    // ...

    // IsysSequence methods. From the point of view of ISYS, a
    // Homolog is representative of the background sequence.
    public String getICAccession () { return backgroundSequence.getICAccession (); }
    public String getSequenceText () { return backgroundSequence.getSequenceText (); }

    // HasTaxon method. Similarly, implemented via background sequence.
    public IsysTaxon getTaxon () { return backgroundSequence.getTaxon (); }
}

```

Fig. 6 Contd...

**Abstract**  
**Introduction**  
**Illustration of ISYS**  
**System architecture**  
**Discussion**  
**Acknowledgements**  
**References**



GO BACK

CLOSE FILE

```

/** Example of event listener that processes visibility events for
 * IsysSequences and IsysTaxons. A listening component would
 * register an instance of this class with the EventChannel (this
 * listener is effectively an interface between components).
 */
class ItemVisibilityListener implements IsysEventListener {
    public void handleEvent (IsysEvent e) {
        Object o = e.getObject (), IsysTaxon t = null, IsysSequence s = null;

        If (o instanceof IsysTaxon)
            t = (IsysTaxon) o;
        else if (o instanceof HasTaxon)
            t = ((HasTaxon) o).getTaxon ();

        If (o instanceof IsysSequence)
            s = (IsysSequence) o;
        else if (o instanceof HasSequence)
            s = ((HasSequence) o).getSequence ();

        if (t != null) {
            // Respond to event in terms of IsysTaxon
        }

        if (s != null) {
            // Respond to event in terms of IsysSequence
        }
    }
}

```

**Fig. 6.** The *IsysSequence*, *IsysTaxon*, and *HasTaxon* interfaces, a simplified version of the *Homolog* class used by the *SimilaritySearcher*, and a sketch of an event listener that might be passed an instance of such a class. The use of Java interfaces allows a *Homolog* object to be interpreted in terms of both its associated *IsysSequence* and *IsysTaxon* objects.

**Abstract**

**Introduction**

**Illustration of ISYS**

**System architecture**

**Discussion**

**Acknowledgements**

**References**



**GO BACK**

**CLOSE FILE**

# Discussion

Our approach to the problem of heterogeneous bioinformatics resources is to enable component integration through a generalized client-side platform, which behaves as both an Event Channel and a service Broker. We have shown with the ISYS prototype that wide-ranging, integrative uses can be supported in this way, even when components and rules for their interaction are simple. However, this approach is not a panacea for the problem of heterogeneity. It does not provide for complex declarative queries, ensure interdatabase consistency, or eliminate syntactic or semantic incompatibilities among databases. Nor does it supersede component development efforts in any way. It is simply an approach to component integration, and thus ultimately depends on the existence of quality components.

Although not presented as an integration solution, the work of the bioWidgets group ([Crabtree et al., 1999](#); [Fischer et al., 1999](#)) prefigures ISYS in several ways. BioWidgets also involves component-based development in Java and its authors also have emphasized the importance of intuitive user interfaces. The main difference is that ISYS strives for complete decentralization, to promote flexibility, configurability, ready incorporation of legacy software, and immediate assembly of systems from components, while bioWidgets accepts the need for top-level controllers, and requires that components be wired directly to one another. The bioWidgets design permits more customized and tighter integration, at the cost of less flexibility and more development time. It is focused on single application developers while ISYS is focused more

[Abstract](#)

[Introduction](#)

[Illustration of ISYS](#)

[System architecture](#)

[Discussion](#)

[Acknowledgements](#)

[References](#)



[GO BACK](#)

[CLOSE FILE](#)

directly on users (in that a user can create an integrated system simply by installing a set of compliant components). ISYS also strives, in a way that bioWidgets does not, to serve as a development platform for integration of separately-developed components. The two approaches are not incompatible: an application built from bioWidgets components could easily be adapted to plug into the ISYS platform, with fine-grained integration (e.g. between panels in a single user interface) occurring in a tight, customized way and coarse-grained integration with separate ISYS components occurring via the ISYS bus. Note that bioWidgets uses JavaBeans but does not use the InfoBus.

The bottom-up philosophy inherent in ISYS is inspired by dynamical systems such as cellular automata and boolean networks (Pagels, 1988; Levy, 1992), in which the interaction of simple, independent agents results in the emergence of surprisingly complex properties. The design challenge has been to find a set of ground rules sufficient to enable interesting interactions among components but not so onerous as to stifle their behavior. With such rules in place one can combine any collection of compliant components, regardless of their particular attributes, and see what happens. Even with the imperfect rules developed so far, the resulting interactions are rich and useful. In addition, systems of such loosely coupled components are more intuitively usable than might be expected. Dynamic Discovery and component synchronization generally produce behaviors that users find natural and sensible.

ISYS differs fundamentally from most database integration solutions as an approach to the problem of heterogeneity. Those solutions generally separate

*Abstract*

*Introduction*

*Illustration of ISYS*

*System architecture*

*Discussion*

*Acknowledgements*

*References*



**GO BACK**

**CLOSE FILE**

data integration from user interface development and take as their main goal support for complex declarative queries, either assuming that sophisticated user interfaces will emerge to help biologists compose queries (Karp, 1996), or attempting to develop such interfaces themselves (Baker *et al.*, 1998). Our experience, however, is that complex queries of irregular biological databases can rarely be specified *a priori*. They are best composed interactively, through a series of simple set-building and set-combining operations, with users having an opportunity to inspect and adjust each intermediate set—as suggested by the Entrez (Sirotkin, 1999) and PathDB (Waugh *et al.*, 2000) query tools. Users need to probe and explore actual data while constructing a query, to discover how to adjust search criteria for inconsistencies and imprecisions. Thus, user interface design and data integration are intertwined.

Our approach is to consider tools for set-building and combination as components that encapsulate details of their interaction with source databases and to expose abstract interfaces only detailed enough for the needs of other components. In this way, the heterogeneous database problem reduces to a component integration problem. The cost of the reduction is the sacrifice of complex declarative queries capable of descending arbitrarily deeply into schematic details and of executing efficiently across multiple repositories. The benefits are system flexibility and maintainability, plug-and-play of data sources, and a natural compatibility with an intuitive, exploratory mode of usage.

Although the ISYS prototype has only very simple set-building tools, the

Abstract

Introduction

Illustration of ISYS

System architecture

Discussion

Acknowledgements

References



GO BACK

CLOSE FILE

platform will support more sophisticated ones. As long as resulting sets can be expressed in the *lingua franca* of the interface-based data model, they can be transferred among components and combined. We believe that most interesting cross-database queries can be supported in this way. Still, data warehouses and multi-database query systems will be better-suited for certain routinely-executed and well-defined queries. ISYS can make use of such systems by fronting them with server proxies, as with any other data server.

The ISYS platform and several components are available in the form of a ‘developer’s kit’ (see *Availability*). Current work includes integration of additional sequence analysis components, coordination with efforts at NCGR in metabolic pathways, gene expression, and genomic maps, and research into general support for batch analysis.

**Abstract**

**Introduction**

**Illustration of ISYS**

**System architecture**

**Discussion**

**Acknowledgements**

**References**



**GO BACK**

**CLOSE FILE**

# Acknowledgements

The authors are grateful to Faye Schilkey, Tom Cartner, David Hanley, and Dawn Perry for contributions in prototype development and documentation, to Allan Dickerman for insights on integration-intensive use cases, to Harry Mangalam for critical reading of the manuscript, and to Stephen Joseph and the NCGR for funding and general support. We also thank several colleagues for helpful feedback on the prototypical system: Ben Bowen, Bill Bruno, Doug Cook, Richard Dixon, Marga Hernandez, Eva Huala, Greg May, Richard Michelmore, Brook Milligan, Dave Neale, Sue Rhee, Lincoln Stein, and Andreas Wagner.

*Abstract*

*Introduction*

*Illustration of ISYS*

*System architecture*

*Discussion*

*Acknowledgements*

*References*



GO BACK

CLOSE FILE

## References

- Aberer,K. and Hemm,K. (1996) A methodology for building a data warehouse in a scientific environment. In *4th International Conference on Cooperative Information Systems*. IEEE Computer Society Press, Brussels, Belgium, pp. 90–101.
- Altschul,S.F., Gish,W., Miller,W., Myers,E.W. and Lipman,D.L. (1990) Basic local alignment search tool. *J. Mol. Biol.*, **215**, 403–410. [MEDLINE Abstract](#)
- Baker,P.G., Brass,A., Bechhofer,S., Goble,C., Paton,N. and Stevens,R. (1998) TAMBIS: transparent access to multiple bioinformatics information sources. In *6th International Conference on Intelligent Systems for Molecular Biology*. AAAI Press, Menlo Park, Montreal, Canada.
- Baker,P.G., Goble,C.A., Bechhofer,S., Paton,N.W., Stevens,R. and Brass,A. (1999) An ontology for bioinformatics applications. *Bioinformatics*, **15**, 510–520. [MEDLINE Abstract](#)
- Benton,D. (2000) Standards to enable bioinformatics data and information integration. In *Barnett International's 2nd Annual Bioinformatics and Data Integration Conference*. Philadelphia, PA.
- Bio-Ontology (2000) <http://www.smi.stanford.edu/projects/bio-ontology/>.
- Boyle,J. (1998) Building component software for the biological sciences. *CCP11 Newsletter*, **4**, 14. [http://www.hgmp.mrc.ac.uk/CCP11/newsletter/vol2\\_2/john/index.html](http://www.hgmp.mrc.ac.uk/CCP11/newsletter/vol2_2/john/index.html)

Abstract

Introduction

Illustration of ISYS

System architecture

Discussion

Acknowledgements

References



GO BACK

CLOSE FILE

- Burge,C. and Karlin,S. (1997) Prediction of complete gene structures in human genomic DNA. *J. Mol. Biol.*, **268**, 78–94. [MEDLINE Abstract](#)
- Buschmann,F., Meunier,R., Rohnert,H., Sommerlad,P. and Stal,M. (1996) *A System of Patterns: Pattern-Oriented Software Architecture*. Wiley, Chinchester, UK.
- Carter,P., Coupaye,T., Kreil,D.P. and Etzold,T. (1999) SRS: analyzing and using data from heterogeneous textual databanks. In Letovsky,S. (ed). *Bioinformatics: Databases and Systems*. Kluwer, Norwell, MA, pp. 213–231.
- Chung,S.Y. and Wong,L. (1999) Kleisli: a new tool for data integration in biology. *Trends Biotechnol.*, **17**, 351–355. [MEDLINE Abstract](#)
- Crabtree,J., Fischer,S., Gibson,M. and Overton,G.C. (1999) Biowidgets: reusable visualization components for bioinformatics. In Letovsky,S. (ed). *Bioinformatics: Databases and Systems*. Kluwer, Norwell, MA, pp. 255–263.
- Davidson,S.B., Overton,C. and Buneman,P. (1995) Challenges in integrating biological data sources. *J. Comput. Biol.*, **2**.
- Davidson,S.B., Buneman,O.P., Crabtree,J., Tannen,V., Overton,G.C. and Wong,L. (1999) BioKleisli: integrating biomedical data and analysis packages. In Letovsky,S. (ed). *Bioinformatics: Databases and Systems*. Kluwer, Norwell, MA, pp. 201–211.
- Etzold,T., Ulyanov,A. and Argos,P. (1996) SRS: information retrieval system for molecular biology data banks. *Meth. Enzymol.*, **266**, 114. [MEDLINE](#)

[Abstract](#)

[Introduction](#)

[Illustration of ISYS](#)

[System architecture](#)

[Discussion](#)

[Acknowledgements](#)

[References](#)



[GO BACK](#)

[CLOSE FILE](#)

## Abstract

- Fischer,S., Crabtree,J., Brunk,B., Gibson,M. and Overton,G.C. (1999) bioWidgets: data interaction components for genomics. *Bioinformatics*, **15**, 837–846. [MEDLINE Abstract](#)
- Fujibuchi,W., Goto,S., Migimatsu,H., Uchiyama,I., Ogiwara,A., Akiyama,Y. and Kanehisa,M. (1997) DBGET/LinkDB: an integrated database retrieval system. *Pacific Symp. Biocomput.*, **3**, 683–694.
- Gamma,E., Helm,R., Johnson,R. and Vlissides,J. (1995) *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, New York.
- Goodman,N., Rozen,S. and Stein,L. (1995) The importance of standards and componentry in meeting the genome informatics challenges of the next five years. In *Second Meeting on the Interconnection of Molecular Biology Databases*. <http://www.ai.sri.com/~pkarp/mimbd/95/abstracts.html>
- Helt,G.A., Lewis,S., Loraine,A.E. and Rubin,G.M. (1998) BioViews: Java-based tools for genomic data visualization. *Genome Res.*, **8**, 291–305. [MEDLINE Abstract](#)
- InfoBus (2000) <http://java.sun.com/beans/infobus>.
- Inman,J. (2000) A distributed system for serving bioinformatics computation. In *Systematics, Cybernetics, and Informatics*.
- JalView (1998) <http://www2.ebi.ac.uk/~michele/jalview/>.
- Jungfer,K., Cameron,G. and Flores,T. (1999) EBI: CORBA and the EBI databases. In Letovsky,S. (ed). *Bioinformatics: Databases and Systems*.

Abstract

Introduction

Illustration of ISYS

System architecture

Discussion

Acknowledgements

References



GO BACK

CLOSE FILE

- Kluwer, Norwell, MA, pp. 245–254.
- Karp,P. (1996) A strategy for database interoperation. *J. Comput. Biol.*, **2**, 573–586.
- Leser,U., Lehrach,H. and Crollius,H.R. (1998) Issues in developing integrated genomic databases and application to the human X chromosome. *Bioinformatics*, **14**, 583–590. [MEDLINE Abstract](#)
- Levy,S. (1992) *Artificial Life: A Report from the Frontier Where Computers Meet Biology*. Random House, New York.
- Life Sciences Research Task Force (1997) of the Object Management Group. <http://www.omg.org/homepages/lsr>.
- Markowitz,V.M. and Ritter,O. (1995) Characterizing heterogeneous molecular biology database systems. *J. Comput. Biol.*, **2**.
- Markowitz,V.M., Chen,I.M.A and Kosky,A. (1996) Exploring heterogeneous molecular biology databases in the context of the object-protocol model. In Suhai,S. (ed). *Theoretical and Computational Genome Research*. Plenum Press, New York, pp. 161–176.
- Markowitz,V.M., Chen,I.M.A., Kosky,A. and Szeto,E. (1999) OPM: object-protocol model data management tools '97. In Letovsky,S. (ed). *Bioinformatics: Databases and Systems*. Kluwer, Norwell, MA, pp. 187–199.
- Monson-Haefel,R. (1999) *Enterprise JavaBeans*. O'Reilly and Associates, Sebastopol, CA.
- NCBI (2000) National Center for Biotechnology Information. Taxonomy

Abstract

Introduction

Illustration of ISYS

System architecture

Discussion

Acknowledgements

References



GO BACK

CLOSE FILE

database and browser. <http://www.ncbi.nlm.nih.gov/Taxonomy/taxonomyhome.html>.

Object Management Group (1996) *CORBA: Architecture and Specification*, OMG publication.

Pagels,H.R. (1988) *Dreams of Reason: The Computer and the Rise of the Sciences of Complexity*. Simon Schuster, New York.

Ritter,O. (1994) The integrated genomic database (IGD). In Suhai,S. (ed). In *Computational Methods in Genome Research*. Plenum Press, New York, pp. 57–73.

Rodriguez-Tome,P. and Lijnzaad,P. (1999) The radiation hybrid database. *Nucleic Acids Res.*, **27**, 115–118. [MEDLINE Abstract](#)

Searls,D. (1995) bioTK: Componentry for genome informatics graphical user interfaces. *Gene*, **163**, GCI–16.

Sirotkin,K. (1999) NCBI: integrated data for molecular biology research. In Letovsky,S. (ed). *Bioinformatics: Databases and Systems*. Kluwer, Norwell, MA, pp. 11–19.

Skupski,M.P., Booker,M., Farmer,A., Harpold,M., Huang,W., Inman,J., Kiphart,D., Kodira,C., Root,S., Schilkey,F., Schwertfeger,J., Siepel,A., Stamper,D., Thayer,N., Thompson,R., Wortman,J., Zhuang,M. and Harger,C. (1999) The Genome Sequence DataBase: toward an integrated functional genomics resource. *Nucleic Acids Res.*, **27**, 35–38. [MEDLINE Abstract](#)

Slidel,T. (1998) The life sciences research task force. *Objects in Bioinformatics* '98. Cambridge, UK.

Abstract

Introduction

Illustration of ISYS

System architecture

Discussion

Acknowledgements

References



GO BACK

CLOSE FILE

- Siepel,A., Tolopko,A., Farmer,A., Steadman,P., Schilkey,F., Perry,B.D. and Beavis,W. (2000) A development platform for the integration of heterogeneous bioinformatics software components. *IBM Syst. J.*, in press.
- Szyperski,C. (1998) *Component Software: Beyond Object-oriented Programming*. Addison-Wesley/Longman, London.
- TAIR (1999) The Arabidopsis Information Resource. <http://www.arabidopsis.org/>.
- Thompson,J.D., Higgins,D.G. and Gibson,T.J. (1994) CLUSTAL W: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice. *Nucleic Acids Res.*, **22**, 4673–4680. [MEDLINE Abstract](#)
- Waugh,M.E., Bulmore,D.L., Farmer,A.D., Gonzales,M., Steadman,P.A., Wlodek,S.T. and Mendes,P. (2000) PathDB: a second generation metabolic database. *Bioinformatics*, <http://www.ncgr.org/research/pathdb> submitted.

Abstract

Introduction

Illustration of ISYS

System architecture

Discussion

Acknowledgements

References



GO BACK

CLOSE FILE