Available online at www.sciencedirect.com
SCIENCE DIRECT®
Journal of Biomedical Informatics 37 (2004) 205–219
Journal of Biomedical Informatics
www.elsevier.com/locate/yjbin

Methodological Review

Bioinformatics integration and agent technology

K.A. Karasavvas,[a,*] R. Baldock,[b] and A. Burger[a,b]

[a] *Mathematical and Computer Sciences, Heriot-Watt University, Edinburgh EH14 4AS, UK*
[b] *Human Genetics Unit, Medical Research Council, Edinburgh EH4 2XU, UK*

Received 10 December 2003
Available online 8 May 2004

Abstract

Vast amounts of life sciences data are scattered around the world in the form of a variety of heterogeneous data sources. The need to be able to co-relate relevant information is fundamental to increase the overall knowledge and understanding of a specific subject. Bioinformaticians aspire to find ways to integrate biological data sources for this purpose and system integration is a very important research topic. The purpose of this paper is to provide an overview of important integration issues that should be considered when designing a bioinformatics integration system. The currently prevailing approach for integration is presented with examples of bioinformatics information systems together with their main characteristics. Here, we introduce agent technology and we argue why it provides an appropriate solution for designing bioinformatics integration systems.

Keywords: Bioinformatics; System integration; Agent technology

1. Bioinformatics system integration

System integration is a challenging research topic, important for most application domains. This is especially true in bioinformatics systems because of the inherent complexity of the domain [1,2] in which: (a) most rules have exceptions; (b) there is a rich variety in data, from one-dimensional genome or protein sequences to three-dimensional models of embryos—3D images demanding vast amounts of storage capacity; (c) complex relationships between structures; (d) variation in curation and quality control standards [3]; (e) multiple sources of similar data, in some cases interpreted versions of the same data [3,4]; and (f) uncertainty, natural variation, experimental error, interpretation error, computational error. In this section we will introduce principal aspects of system integration with a focus on bioinformatics systems. Issues like heterogeneity, federated systems, and 'wrapping' legacy systems will be discussed with examples of bioinformatics integration systems.

1.1. Fundamental aspects of integration

The main goal of integration is to provide mechanisms that can unify a number of (computer) systems. By systems we mainly refer to data sources, like databases, web servers, and so on. Instead of having to manually request (query) data from various resources and then combine the results to get more useful information, one would like an integrated system that can automate such a process. We can describe such functionality as a number of steps: (a) the user makes a request (query) to the integrated system; such a request may require more than one data source to be satisfied; (b) the integration system processes the request and decides how to split it into sub-requests specific to data sources; (c) the sub-requests are made and all individual results are returned to the integration system; and (d) the results are combined to a coherent answer which is returned to the user. Three important aspects of system integration are distribution, autonomy and heterogeneity [5,6].

* Corresponding author. Fax: +44-131-451-3249.
E-mail addresses: ceekk@macs.hw.ac.uk (K.A. Karasavvas), R.Baldock@hgu.mrc.ac.uk (R. Baldock), A.Burger@hgu.mrc.ac.uk (A. Burger).

1532-0464/$ - see front matter © 2004 Elsevier Inc. All rights reserved.
doi:10.1016/j.jbi.2004.04.003

Distribution. In most cases data sources are distributed. The user need not know the location and other details of each available resource. Such details are usually transparent and handled automatically by the integrated system. Distribution should preferably be hidden from the user.

Autonomy. It is very often the case that integrated resources belong to different organisations or research groups. While most people are willing to share their data, they do not want to lose control over decisions for their data source. Thus, the developers of an integrated system do not usually have any control over the underlying systems, which are autonomous.

Heterogeneity. In an open and diverse environment it is very common that some or all of the data sources are different from each other. Integrating heterogeneous systems involves extra work so as to ensure the correct relationship of data between the information systems.

1.2. Heterogeneity

Heterogeneity of data sources has been the focus of many studies in the past [5–9]. While there are some differences in how each study analyses heterogeneity we can identify two major categories:

Technical. Such differences can occur because of different hardware platforms, operating systems, database management systems (query languages, data models), access protocols, transport formats, and programming languages.

Semantic. Conceptual differences occur in the data models/schemas[1] of the data sources [10,11], i.e., the organisation of data and the relationships between such data. Typical examples are *synonyms*—when attributes of two schemas have different names but refer to the same concept—and *homonyms*—when attributes of two schemas have the same name but refer to different concepts.

In general, technical heterogeneity is easier to resolve. The Java programming language can be used to deal with different hardware and operating systems, common query languages, like SQL, could be used to deal with database management systems, Web technologies, like HTTP and XML, can be used for common access and formatting, respectively, and CORBA [12] to deal (among others) with different programming languages.

An important and challenging aspect is to integrate the schemas of the data sources. Different schemas and data models can introduce both technical, e.g., relational versus object models, and semantic heterogeneity. To bridge schema heterogeneity we usually define a *common schema* expressed in a common data model (CDM). Each local data model is mapped to the CDM thereby resolving semantic heterogeneity. Integrated systems that aim to create a CDM and a federated schema are called *federated systems* [5,6].

1.3. Federated systems

Federated systems can be classified in terms of their degree of federation and instantiation [6,13]. The first refers to how autonomous—independent from the integration system—the data sources are; autonomy indirectly influences the precision of the schema integration. We can have a tight federation which typically involves non-autonomous data sources—potentially very precise matching of the local schemas—, and capability to allow reliable read–write access to the integrated system. Alternatively, a loose federation typically means completely autonomous data sources—constraint matching of the local schemas—, and only read-only reliable access to the data sources. Usually, when dealing with an open environment with a large number of data sources, a loose federation is the preferred option because it is easier to extend [14].

The second, the degree of instantiation, refers to where the physical data reside. We can have a virtual federation which means that the actual data reside in the respective data sources, and the integration system provides just a unified view of these data sources. Or a materialised federation—also called *warehousing*—in which the integrated system consists of a global physical repository which includes all the data sources' data. Although a materialised solution is more efficient computationally, in general the virtual approach is preferred as it does not involve data replication—which introduces data update and synchronisation problems—and it is much easier to maintain [13].

1.4. Legacy systems and wrappers

Typically, each data source has a query language (or sometimes an API) that allows users to request data from that resource. This query language is designed with the data model in mind so as to achieve a more natural mapping between the two. To deal with query language heterogeneity, integration systems use a global query language—also called internal or common query language (CQL). This language is used as the common language between the heterogeneous data sources and it should be designed according to the common data model used, i.e., facilitate the appropriate expressiveness.

With a query formulated in the CQL the integrated system could use the federated schema to decompose, usually referred to as *query decomposition and planning*, the initial query to sub-queries that could be answered

[1] When we talk about the schema of a data source we refer to the conceptual data model—the conceptual relationships between the data. A schema is expressed in the data model of each particular data source [6].

by individual resources. The sub-query, expressed using the CQL, is then translated to the data source-specific language. This task is accomplished by using software modules called *wrappers*. Wrappers encapsulate or 'wrap' the functionality of existing legacy systems. They are responsible for converting a request formulated in the CQL to the specific query language used by a data source and vice versa. As can be seen in Fig. 1 wrappers can make any kind of data appear homogeneous to the integration system that only 'understands' the CQL.

1.5. Mediation and bioinformatics integration systems

The previous discussion of integration implicitly described one of the most common integration approaches in bioinformatics: mediation [15,16]. Mediators were introduced with the argument that they "simplify, abstract, reduce, merge, and explain data" [15] and their primary purpose is seamless integration of heterogeneous data sources. A general definition given by Wiederhold [15] is: "A Mediator is a software module that exploits encoded knowledge about certain sets or subsets of data to create information for a higher layer of applications." Mediation is an abstract architecture that conceptualises integration. In our integration overview we presented a more practical view of the integration procedure that is summarised in Fig. 2.

We can now describe the integration steps in more detail:

- The user provides a query formulated in the common query language to the integration system—mediator(s).
- The integration system applies the query to the common data model. The query decomposition and planning module (part of the mediators) decomposes the initial query into sub-queries, again formulated in

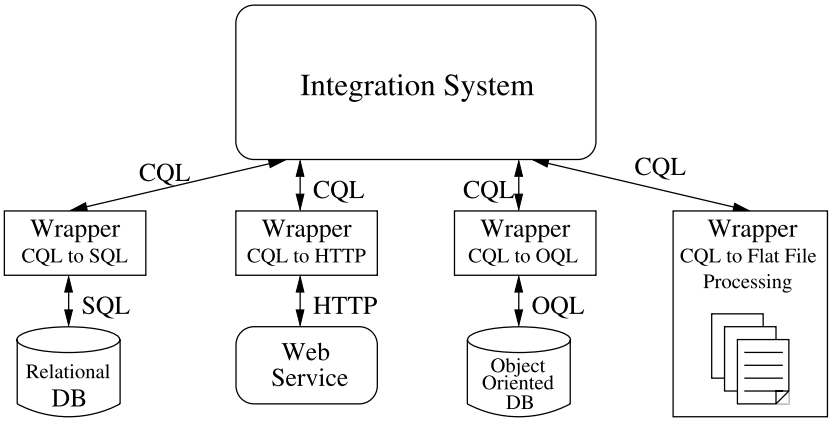

Fig. 1. Example of wrappers translating the common language (CQL) to the local query languages.

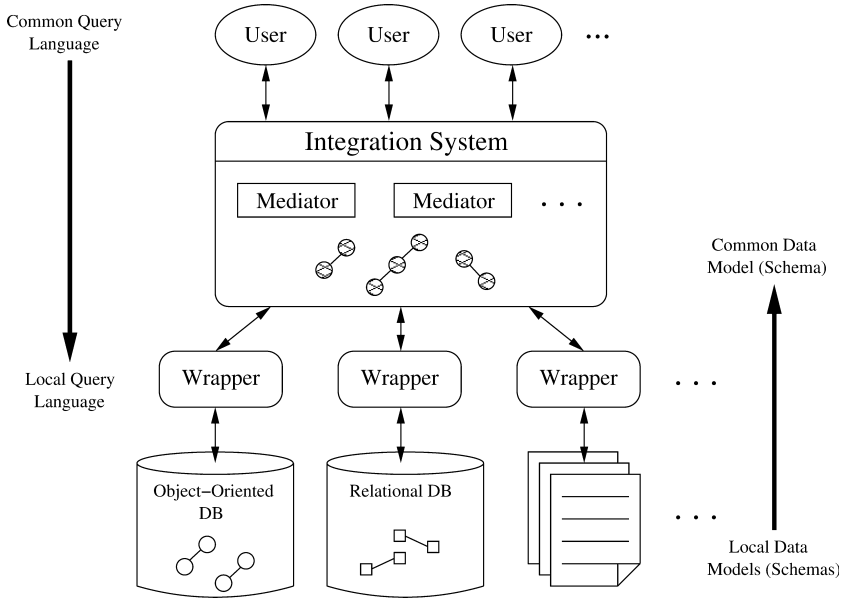

Fig. 2. Overview of integration procedure.

the common query language. The sub-queries are passed to the appropriate data sources via their respective wrappers.
- Each wrapper translates the sub-queries to the local query language used by the data source and then translates the results back to the CQL.
- The results are then returned to the integration system where they are combined to a coherent result, which is returned back to the user.

Systems that do not provide a conceptual model in their CDM, or not a CDM at all, cannot provide integration and location transparency. That means that the user has to define how the data sources' data will be combined and which data sources should be used; we refer to such systems as *non-transparent*.

Bioinformatics integration systems follow the procedure illustrated in Fig. 2 and their functionality can be generally described with the integration steps mentioned above. Table 1 provides an overview of these systems, presenting aspects of their federation, their common query language (CQL) and common data model (CDM) as well as whether they provide total transparency to the users or not. Some of the systems in Table 1 provide potential variations of the mediation approach to integration and these are briefly described below.

In BioKleisli [17], when users formulate queries in the Collection Programming Language (CPL) [18] they have to provide information of which data sources will be used and how. Thus, the actual integration of the resource's data is manual and takes place while formulating the query. BioKleisli does not provide a CDM. The query itself has the integration and location details, which means that integration and distribution are not transparent to the user. Wrappers—called Data Drivers in BioKleisli—have been developed for many data sources, including GenBank, EMBL, DDBJ, GSDB, GDB, and BLAST.

TAMBIS [19] uses GRAIL [20], a description logic [21], to capture the 'Concept Model' (CDM). The location details and semantic heterogeneity are captured in the 'Source Model.' A user-interface driven by the 'Concept Model' allows users to construct queries in GRAIL. Then, the system translates these queries, with the help of the 'Source Model,' into a CPL query. Finally, the CPL query can be executed using a BioKleisli-like system, which in effect will act as a (global) wrapper.

K2 [22] takes an object-oriented approach to mediation. Its CDM is a hybrid of ODMG-standards [23] Object Definition Language (ODL)—to define data objects and their relationships—and Object Query Language (OQL)—to define the location details of the data. Additionally, OQL is used as the common query language. K2 aims to provide the tools needed to build an integration system, using the mediation approach and the object model. Some examples of data sources connected to K2 are SRS, KEGG, BLAST, Genomes, USPTO, Delphion, PubMed, and MMDB.

'OPM Tools' [24] provide tools for mediation with another object model. Object-Protocol Model (OPM) and OPM*QL are similar to ODL and OQL, respectively. 'OPM Tools' have been used to create a database federation that includes GDB, GSDB, and GenBank.

P/FDM [25] uses PROLOG as a CQL to access databases using the Functional Data Model (FDM). In addition, a prototype system was built to accept queries in a higher-level language, called Daplex—a language similar to OQL. A query in Daplex can generate PROLOG code to access databases using the FDM, generate SRS-QL code to access SRS, or both.

The Sequence Retrieval System (SRS) [26] combines both virtual and materialised federation. It treats data sources as a series of entries that are defined in the Icarus language. Indices of the entries of the data sources are kept in a global repository, while the actual data are kept in their respective data sources. Note that with Icarus one can only represent the entries of the data sources; there is no conceptual model (schema) and like BioKleisli the integration and location details are not transparent to the user.

The Integrated Genomic Database (IGD) [27] uses the Concise Object Query Language (COQL) and an Extended Entity Relation (EER) model for the CQL and CDM, respectively. One of the few systems that uses materialised federation, IGD, integrates more than a

Table 1
Bioinformatics integration systems

Integration system	Federation	Instantiation	CQL	CDM	Transparency
BioKleisli	Loose	Virtual	CPL	—	No
TAMBIS	Loose	Virtual	GRAIL	'Concept Model' & 'Source Model'	Yes
K2	Loose	Virtual	OQL	K2MDL	Yes
OPM Tools	Loose	Virtual	OPM*QL	OPM	Yes
P/FDM	Loose	Virtual	PROLOG, Daplex	FDM	Yes
SRS	Loose	Virtual & Materialised	SRS-QL	Icarus	No
IGD	Tight	Materialised	COQL	EER	Yes
DiscoveryLink	Loose	Virtual	SQL	SQL DDL	Yes

dozen data sources quite successfully, although its extensibility has been questioned [13].

Finally, DiscoveryLink [28] from IBM uses SQL for a CQL using wrappers to map non-relational data (flat text files, object model, etc.) to a relational model. Wrappers provide a description of the data source that they represent using an SQL Data Definition Language (DDL)—the CDM. Unfortunately, DiscoveryLink has the accepted disadvantage of only being able to handle limited semantic heterogeneity. Having said that, DiscoveryLink is based on well-tested IBM technologies like DataJoiner and DB2 Universal Database, as well as non-relational wrapper technology from the Garlic project [29], offering an industrial strength solution.

All of the above bioinformatics integration systems follow—each to a different extent—the system integration procedure that is prevalent to computer science; by providing a CDM and a CQL as an intermediate layer, an integration system can dynamically answer any queries related to the integrated data sources—as described by the CDM.

1.6. Confidence in results' quality

In previous sections, we have seen that integration could be either transparent to the user or not. Transparency avoids the need for the user to be an expert in bioinformatics systems and databases; i.e., to know which data sources contain the information needed and how the resulting data should be combined to reach a final result. A transparent system incorporates integration reasoning, put there by bioinformatics experts, and automates the integration procedure.

However in many cases there is more than one way to solve a particular problem. For example, what scoring matrix should be used for sequence comparison, and what database should be used to find tissue-specific gene expression data. Such decisions are not straightforward, and in most cases it cannot be part of the integration logic because it really depends on the user: what experiment (s)he is involved with, how important are the specific data to obtaining the final result, and so on.

Thus, it would be convenient for the user to be able to intervene at the level of integration logic. However, allowing the user to intervene limits the system's transparency. Consequently, we have identified the need for *adjustable transparency* instead of the fixed transparent or non-transparent. Although, integration would be transparent the user should be able to adjust the level of transparency according to his/her needs. Work on this area is limited in bioinformatics integration systems, and we believe it is an important issue that should be dealt with to build large and more content-rich systems.

1.7. Semantic web services, semantic grid, and integration

Many bioinformatics data sources are becoming publicly available through the Internet and more recently as Web services. This has the benefits that it enables developers to publish, as well as locate services on the Web, so that anyone can access them in a uniform way. However, as with data integration, developers could not be certain of the purpose (semantics) of the service, e.g., does the concept 'price' include VAT or not? The *Semantic Web* have been developed to provide a solution. The goal is to provide common meaning between concepts used in Web pages and services. To this end: (a) a general-purpose data format has been designed (XML—see [30] for using XML in bioinformatics data integration); (b) it was extended to allow for metadata (RDF); (c) basic semantics for the data structures and values allowed have been specified (RDF Schemas), and recently; (d) fully developed *ontology* languages have been defined, e.g., the Web Ontology Language (OWL).

An ontology [31,32] is a group of concept definitions that describe an application domain. As mentioned, knowledge terms can have a different meaning between different application domains or even between different researchers in the same domain (a representative example in biology is the different conceptualisations used for 'gene' [2]). That is why we need *common ontologies*. Disparate researchers and/or systems can commit to a common ontology to accomplish the same understanding for a set of concepts.

The very large-scale distributed computing and data required of particle physics coupled with the need to go beyond the limited stateless and insecure Web Service technology has led to the development of the Grid [33]. The goal was to inter-connect a large amount of computing resources at a national or even world-wide scale to build 'cheap' virtual supercomputers. Other research communities, such as biology, earth science, and astronomy, have expressed interest in the Grid. This change of focus made other extensions necessary; for example, to resolve heterogeneity of the disparate resources and to incorporate ontologies, led to what is now called Semantic Grid. There is now a large-scale effort to develop Grid standards and technology under the OGSA/OGSI [34] heading with many national and international Grids now under development.

In the bioinformatics literature, in the context of the Semantic Web and Grid, 'system integration' is also used in a more general sense,[2] i.e., that of mustering a large number of data sources and providing a framework for their discovery and execution. Two such notable systems are BioMOBY [35]—for bioinformatics

[2] In contrast to what we discussed in previous sections.

Web Services—and myGRID [36]—for a bioinformatics Semantic Grid.

BioMOBY describes bioinformatics services as Web Services. It provides a language to describe biological services in terms of their inputs and outputs, as well as a central registry, called 'MOBY Central,' to enable service registration and discovery.

MyGRID provides similar functionality and is based on the Open Grid Services Architecture (OGSA) [34]. In addition, it offers the users the ability to use or create their own *workflows*. A workflow is a sequence of services executed in the correct order that combined can provide higher-level services. Workflows can be thought of as pre-generated static plans, as opposed to the (query) plans generated dynamically by mediators.

1.8. Investigating agent technology

Because of the the huge diversity and data volume in the biology domain there already exists a significant amount of ontologies [37,38]. Many projects and conferences have been dedicated to further work on this area and in 1998 a consortium was formed, comprising collaborations between many bioinformatics data sources' curators, called Gene Ontology (GO) [39].

Agent technology is a rapidly evolving interdisciplinary field in computer science that emphasises the use of autonomous software entities with the ability to interoperate with other such software entities, in a uniform and standardised way. Because semantic heterogeneity is a fundamental part of interoperability, agent systems used ontologies from the beginning. In the following sections we introduce agent technology and argue that it is appropriate for bioinformatics systems integration.

There are two primary reasons why agent systems could be ideal. Firstly, biology's ontological work could exploit the potential of agent technology, in relation to semantic heterogeneity, more easily. Secondly, it has been argued [40] that "agent-oriented approaches are well suited for developing complex, distributed systems," which applies to bioinformatics integration systems. Agent technology has been successfully applied in the past to system integration [29,41–43]. However, in bioinformatics systems it has mainly been used for enhanced automation [44–49] and thus far only a couple of bioinformatics integration systems are based on agent technology [50,51], and even these do not provide adequate arguments for their use and applicability. In the next section, we will present basic aspects of agent technology. The aim is for the reader to get acquainted with the technology so that (s)he can appreciate the benefits that agents could offer to bioinformatics integration systems. After each section describes some aspect of agent technology we will discuss the implications of this aspect to integrating bioinformatics systems.

2. Agent technology

Software agents aim to provide enhanced automation to information systems. Their main focus is to perform certain tasks on behalf of the user. The Oxford English dictionary defines the word *agent* as "a person who acts for another in business, politics, etc. (estate agent; insurance agent)." However, in computer science, there is no consensus in the research community about the definition of an agent. And there may not be one in the future either, which was also the case with other terms like *object* or *artificial intelligence* (AI).

Despite the minor confusion, the role of agents in computer science is becoming clearer with time. And although a universal definition might never be accepted, the role, use, advantages, and disadvantages are becoming more accepted and established—similar to AI and object-oriented programming.

Agent technology has its roots in multiple research areas including distributed systems, AI and social (e.g., agent organisations and communication) and economic (e.g., auction protocols) sciences. Each individual researcher defines agents according to his/her background and perspective and that creates important diversification to the definitions. A large number of agent definitions can be found in the literature, Franklin and Graesser [52] provide a review, and reach yet another definition. While, we will not go into the debate of which definition is better, we can notice that the common characteristic attributed to an agent is its *autonomy*—its ability to exercise control over its internal state and actions without direct human or other intervention. Some other agent properties can be seen in Table 2. These properties can be used to classify agent systems. Two well-known taxonomies are Nwana's [53] and Franklin and Graesser's [52].

Working in isolation makes software agents depend only on some kind of user feedback to obtain new input. The ability to exchange information with other agents enables them to work together—sharing their knowledge—to achieve a common goal. One that no single agent could solve by itself. Such systems with a number of co-operating agents are called multi-agent systems (MAS).

Agents[3] that are part of a MAS need to possess another property—except autonomy—that is considered fundamental for such systems: communication. Proper communication can be achieved by using a high-level language independent of any computer hardware or operating system. Such an Agent Communication Language (ACL) can be thought of as the equivalent of natural languages used by humans. In effect multi-agent

[3] From now on, when we talk about agents we will always mean agents that are part of a multi-agent system. Similarly, when we talk about an agent system we will mean a multi-agent system.

Table 2
Some agents' properties

Property	Description
Autonomous	agents can exercise control over their internal state and actions without direct human or other interaction
Communicative	are sociably able
Reactive	respond in a timely fashion to their input
Pro-active	are goal-oriented and take the initiative where appropriate
Planning	can plan their own actions
Persistent	have temporally continuous state
Adaptive	can learn and change their behaviour on the basis of their previous experience
Mobile	have the ability to transport themselves from one machine to another

systems try to simulate our society. Each agent takes a particular role (task specialisation) and talks to its peers when in need of a task that it is not capable of carrying out by itself or to notify them that certain conditions have changed.

Fig. 3 is a simple example of a MAS for accessing biological data. The 'User Agent' is responsible for accepting human queries/input/preferences. As an example we will use the query: "find mouse tissues that express the input genes in a given developmental stage (e.g., Theiler stage)." The 'User Agent' then asks the 'Tissues Agent' to find tissues (its role/specialty) providing a set of genes, an organism and a developmental stage. The 'Tissues Agent,' in its part, consults appropriate agents—capable to answer such a query, e.g., 'GXD Agent' and 'EMAGE Agent'—to acquire tissues from the respective data sources, GXD [54] and EMAGE [55]. The results are received by the 'Tissues Agent,' which merges all the tissues returned on the specified developmental stage. Then it returns the resulting set of tissues to the 'User Agent,' which in turn presents them to the user.

Central to agents' functionality is a common communication language, which enables them to co-operate. ACLs are described below.

2.1. Agent communication languages

Communication between agents is modelled as the exchange of declarative statements, typically based on first-order predicate logic.

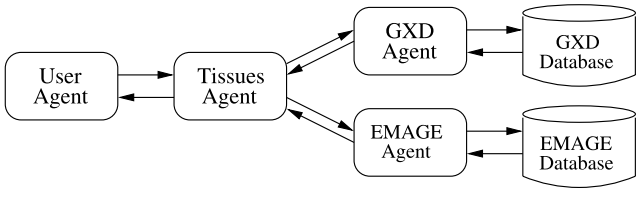

Fig. 3. A simple multi-agent system.

Continuing the parallelism that a MAS simulates a society for agents, we can go further by examining language. Linguists describe certain characteristics [56] of natural languages, which can be considered as the different parts needed to properly describe a language. These characteristics are phonetics, phonology, morphology, lexis (words), semantics, syntax, and pragmatics. The first three describe the oral aspects of communication and thus we can ignore for our purposes. The last four need to be retained. By including lexis in the semantics we can use this sub-set of natural language characteristics to describe a language for agents. To be more precise it can be partitioned into three layers:

Pragmatics. Specifies the way that an entity will express its needs or/and the effect that it wants to pass to the receiver. This layer can be thought of as the specification for information exchange. It specifies the way that two (or more) entities—agents—will communicate and it comprises, among other things, sender, recipient, and the content of the message. Pragmatics is referred to as the Agent Communication Language or ACL. In natural languages, pragmatics are implicit—the intent of the speaker is inferred by his intonation and choice of words—whilst with agents we need to be explicit using specific verbs (speech acts [57,58]), which denote the nature of the communication. Examples of such a specification are FIPA ACL [59] in which speech acts are called *communicative acts*, and KQML [60] in which speech acts are called *performatives*.

Syntax. Used to structure the information that will be sent (i.e., message content). The content of the message contains words that are arranged according to a structure, defined by the syntax of the language. Examples of such languages include FIPA SL [61], KIF [62] or PROLOG [63].

Semantics. A correctly structured message content consists of a number of words (lexis). However, these words do not mean anything to computational entities such as agents. They are just strings that make up a larger string, the message. Semantics is used to give meaning to these words. It ensures that the word is associated with the correct concept. By doing this we can avoid inconsistencies such as having different words for the same concept or one word for different concepts. A group of concept definitions describing a specific domain is called an ontology. Semantics for ACLs can comprise of multiple ontologies.

This layered approach helps us to work on each one part of the language independently. For example we could use a different syntax to describe the information that we want to send, without needing to change the pragmatics and semantics of the language. An example of an ACL message—using the FIPA ACL specification format—can be seen in Fig. 4. FIPA specifications provide both formal and informal definitions for all the

```
(inform
        :sender   AgentA
        :receiver AgentB
        :content  expressed(msx2, brain).
        :language PROLOG
        :ontology biology
)
```

Fig. 4. An example of a FIPA message. AgentA informs AgentB that gene 'msx2' is expressed in tissue 'brain.' The message is expressed in 'PROLOG' and the words used in the message comply with a biological ontology, 'biology.' That means that the receiver agent should understand (or be able to learn/translate) the 'PROLOG' language and the 'biology' ontology.

communication terms used in the ACL messages exchanged, i.e., what are the speech acts (like 'inform'), what is their semantic meaning, what kind of expressiveness does a content language need to provide, and so on.

2.2. ACLs and bioinformatics integration

Note that the exchange of information plays a central role to a MAS; similarly to system integration. The three-layered approach for communicating a message is a big step towards resolving heterogeneity—which was after all one of the main goals of MASs (see next section). More specifically:

- a common ACL with a pre-specified content language takes care of any potential technical heterogeneity as it provides a common intermediate representation of the exchanged data and
- a common ontology resolves any potential semantic heterogeneity.

Additionally, as the purpose of an ACL is the communication between agents, one can think of it as the CQL of an integration system. That has important implications in using agents for integration as they provide a natural framework to achieve interoperability across heterogeneous data and computational sources.

2.3. KSE and FIPA

Around 1990 the Defence Advanced Research Projects Agency (DARPA), understanding the need for information sharing, initiated the Knowledge Sharing Effort (KSE) [64]. Its goal was to develop techniques, methodologies, and software tools for knowledge sharing and reuse at the design, implementation, and execution stages. At that time the term 'agent' did not exist—at least not with the meaning that we attribute to it today. The KSE model was intended for information exchange between databases, expert systems and any other system that could be viewed as a virtual knowledge base. Nonetheless, the main focus was to share information, which implies communication, which in turn implies a common language for communication.

This led to the concept of an ACL, as we use it today, and provided KQML [60] as the means of communicating information. KSE first introduced the layered approach of communication specifically by analogy to natural languages. The KSE model consisted of KQML, KIF [62], and Ontolingua [65] for the pragmatic, syntactic, and semantic layers, respectively.

KQML and communicating agents became very popular and soon there were a number of KQML variants each addressing different aspects and shortcomings noticed during KQML's usage. A number of systems were developed from various academic and commercial institutions. The popularity of KQML increased even further but the diversity of the KQML implementations made it impossible for different systems to interoperate.

Major international companies realised that to promote agent technology to the market it was essential to achieve interoperability between agent systems from different vendors. For this reason, in 1996, these companies joined to form a forum, the Foundation for Intelligent Physical Agents (FIPA) [66], to discuss, design, and provide specifications for agent technology. FIPA's mission statement is: "FIPA is an international organization that is dedicated to promoting the industry of intelligent agents by openly developing specifications supporting interoperability among agents and agent-based applications."

FIPA took advantage of the experience provided from all the KQML variants, identifying shortcomings and dealing with them in its specifications. From the early stages of the FIPA specifications, its designers took great effort to provide unambiguous meaning for the communication between agents—to maximise interoperability. For that reason FIPA provides both formal and informal definitions for all the communication terms used in the ACL messages exchanged. In addition, similar definitions should exist for the language selected for the message content.

That was an organised and centralised effort to create standards with the appropriate commercial support. In December 2002, FIPA have promoted a number of its specifications to standards. With the number of companies adopting the FIPA standards and the lack of competing proposals in agent interoperability it seems that FIPA succeeded in establishing its specifications as the accepted international standards in MAS interoperability.

2.4. Standardisation and bioinformatics integration

It has been argued that a major part of the integration problem in bioinformatics is not technological but sociological [2]. That is, in the absence of a standardised

interface it is very difficult for data source providers to agree on a common way to provide services. Data source providers prefer to be autonomous and usually choose the more convenient methods to represent and offer services rather than those that will help decrease the heterogeneity issue. Of course, that is understandable as each provider will have different needs.

Ideally, each data source provider would provide an interface that complies to a standard. It is here that the agent interoperability standardisation efforts can be of great use. By embracing the FIPA standards data providers can just implement a FIPA-compliant agent that provides an interface to their data source. Automatically, all other FIPA-compliant agents—that understand the content language and ontologies used—will be able to acquire the data offered.

2.5. Planning in multi-agent systems

Traditional centralised planning [67,68] is not enough for a system that is naturally distributed. In this section we will briefly describe the most common strategies in distributed problem-solving.

One of the most classic and popular techniques to distribute problem-solving is by *task sharing*, also called *task passing*. The idea is straightforward. Each agent tries to solve the given problem and when it reaches a task that it does not know how to handle it requests help from other agents. The basic steps in task sharing are [69]:

Task decomposition. Generate a set of tasks to be passed to other agents. This could generally involve decomposing large tasks to sub-tasks that could be tackled by different agents.

Task allocation. Request from the appropriate agents to handle the sub-tasks.

Task accomplishment. The appropriate agents each accomplish their sub-tasks—which may require further task decomposition and allocation.

Result synthesis. When an agent completes a sub-task that it was responsible, it sends the result back to the requesting agent. The last will then synthesise the results into a solution, which could be a sub-solution and thus, in turn, needs to return the result to its requesting agent, until we reach the initial (root) agent that will compose an overall solution.

Notice how similar the above steps are to the typical integration procedure described in Section 1.5—a query is one type of task. Multi-agent task sharing is naturally capable of dealing with mediator-like integration problems.

In task sharing each agent makes a local plan (centralised planning) and then requests, in a way, other agents to continue part of the planning—by solving sub-tasks of the same problem. Hence, globally, the planning process—and execution—is distributed and potentially in parallel. In Fig. 5 each one of the agents depicted acts as a planner—using traditional centralised planning—and co-operates with the rest to achieve a common goal. Thus, the overall plan is distributed among these agents and the process of planning and execution is managed dynamically and in an incremental way. Moreover, when a task is decomposed to, say, t_1 and t_2 in 'Agent 1' and the two sub-tasks are independent of each other[4] then they can also be executed in parallel—which implies that the two agents that receive these sub-tasks, 'Agents 2 and 3,' respectively, will also plan (and execute) in parallel. The agent that initially decomposed a task acts as a synchronisation point for the parallel execution of the sub-tasks, e.g., parallel sub-tasks t_1 and t_2 are synchronised in 'Agent 1.'

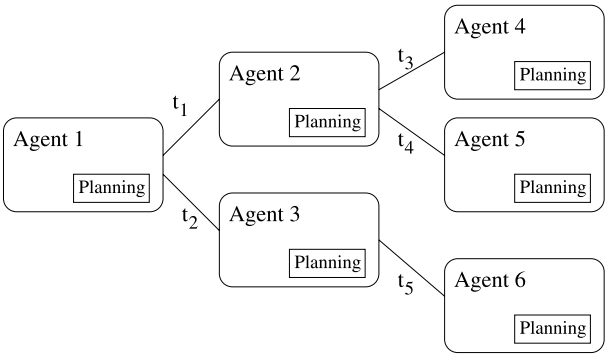

Fig. 5. Example of multi-agent task sharing problem-solving.

Other types of distributed planning [69,70, chapter 8] are: 'Centralised Planning for Distributed Plans,' 'Distributed Planning for Centralised Plans,' and 'Distributed Planning for Distributed Plans.'

2.6. Planning and integration

The steps of task sharing are very similar in structure to the integration steps described in Section 1.5. If we consider a task to be a query, as expressed in the CQL, then the two procedures are almost identical. This implies that significant synthesis is possible between the technologies developed for these complementary activities.

2.7. Adjustable autonomy

As previously discussed, it is widely accepted that one of the primary characteristics of agency is *autonomy*. One fundamental implication of this is that autonomous agents have total control over their behaviour. Although such control is usually helpful, because it indicates automation, users find it difficult to completely trust

[4] Sub-task t_2 does not need the partial result returned from sub-task t_1 and vice versa.

autonomous agents [71] because of possible loss of control, and feeling that the system limits their usual behaviour constraining them from a free choice, e.g., a user might prefer a specific method for solving a task but the autonomous agent chooses another.

To remedy this lack of trust and to increase user control, some systems provide support for *adjustable autonomy*.

> Adjustable autonomy means dynamically adjusting the level of autonomy of an agent depending on the situation. For real-world teaming between humans and autonomous agents, the desired or optimal level of control may vary over time. Hence, effective autonomous agents will support adjustable autonomy. [72]

In other words, it is beneficial to have a mechanism that enables control over the behaviour of a dynamic and complex distributed system so as not to feel uncertain about the quality of the results. This is a research area that is attracting more attention as the MAS become larger and more complex, which in turn causes more work to be focused on the subject [73,74].

2.8. Adjustable transparency

As we have seen, in open bioinformatics integration systems, the issue of the results' quality (see Section 1.6) is even more important to the user. We already suggested that we need a way to adjust the integration system's transparency. Adjustable autonomy has as a result for the user to gain some control over the behaviour of the agents. If the agents' functionality is to integrate then adjusting their autonomy is exactly what we need to better manage the system's transparency [50].

2.9. Agents and software engineering

Software engineers always try to find new methods to make software design easier. Programming structures, procedures, and objects are all part of the engineering effort to abstract and conceptualise the software design process. This aids the design and development of more complex software systems. A successful demonstration for handling complexity was object-oriented programming. It has been argued that agent-based systems are a step further to that end [40,75].

Techniques identified to handle software complexity are *decomposition*, *abstraction*, and *hierarchy* [76]. We will examine these techniques in relation to both objects and agents.

Decomposition. We divide a large problem to smaller problems that are more manageable. The designer can then focus his attention on each sub-problem in relative isolation. Both objects and agents are capable of decomposition—they both facilitate for modular design (but see 'Abstraction').

Abstraction. We define a simplified model of the system so that we emphasise certain important aspects/properties while suppressing others. The designer can then focus his attention on a smaller number of system characteristics, which are usually also high-level. Objects represent a fine granularity of abstracting functionality, though combined they can represent larger entities. Agents abstract functionality of the system at a higher level, that of roles. Each agent would be responsible for a specific task (role) in the same way that individuals in a company are responsible for a specific job.

Hierarchy. We define and manage the relationships between the various problem-solving modules. This allows the designer to combine the functionality of the modules more effectively increasing the useful operation of the modules, e.g., grouping modules and treating them as a higher-level module. In the case of objects, method invocation, is the only mechanism available for describing the interactions that might take place. On the other hand, agents facilitate the more advanced mechanism of an ACL which allows the different modules (agents) a wide variety of interactions.

We can see that objects and agents have many similarities—see [77, pp. 34–36] for a comparison. Agent technology provides the designer with an intuitive approach to capture system functionality at a higher level, while providing for a modular design and a more flexible interaction mechanism. Although agents themselves will possibly be designed using object-oriented techniques anyway, they naturally provide a stronger notion for organisation and co-operation, which in turn provides solid foundations for dealing with complex distributed systems.

2.10. Bioinformatics and complex distributed systems

Most bioinformatics integration systems follow the virtual approach to federation. In addition, the biological domain exhibits many complexities (as discussed in Section 1), which imply that bioinformatics integration systems can be considered as complex distributed systems. We have discussed the advantages that agents have in software engineering, especially when we are dealing with complex systems (see [40] for a detailed discussion), thus arguing that agent technology would be a good choice to deal with the intricacies of bioinformatics integration systems.

3. Agents and integration

Most agent integration systems use the mediation approach [29,41–43]. An agent (or a number of agents) acts as a mediator, usually called Mediator Agent (MA). This agent has access to the CDM, which could be represented using any representation language—including

the content language of the ACL. Additionally, a number of agents will act as wrappers, usually called Resource Agents. We have already discussed that the ACL could be used as the CQL, which is convenient but not restrictive—any CQL could be used.

3.1. Distribution, autonomy, and heterogeneity

Agents have been designed with the intention of information exchange between data sources. Sharing information is a major part of system integration, and thus agents naturally cover the fundamental aspects of integration (discussed in Section 1.1).

Distribution. A MAS is naturally distributed. Multi-agent systems offer location transparency by providing facilities for service discovery and brokering. In addition, a high-level communication language enables flexible and advanced communication between distributed agents.

Autonomy. Agents are designed with the assumption that software entities are autonomous. Moreover, FIPA's interoperability standardisation efforts will ensure communication between agents created from many different vendors, organisations, or research groups. Thus, for example, each data source curator could create resource agents that 'wrap' their data source, enabling other users or (integration) systems to communicate and make use of them. That potentially solves a big part of the sociological problem.

Heterogeneity. Agents communicate only via an ACL. A common communication language and a common message content language deal with technical heterogeneity while sharing an ontology handles the semantic differences. Of course, that does not mean that every agent will understand all possible languages and/or ontologies that could potentially be used. However, here is where Ontology Agents [78] come to great use, as they can translate between a potential language and/or ontology a message uses and a language and/or ontology that a particular agent understands.

3.2. Legacy systems and wrappers

There are a wide number of non-agent software services that could be utilised by an agent system. Non-agent systems could be made a part of an agent community if they were able to communicate using an ACL. In effect the ACL acts as the CQL, with wrappers translating from the ACL to the local query language (or any other kind of interface) and vice versa.

Fig. 6 shows three possible ways to do this—called *agentification* [79] process. A Transducer is an agent that knows how to translate requests from an agent system—other agents—to the non-agent system's interface and vice versa. The advantage of this method is that we do not need access to code and only the communication

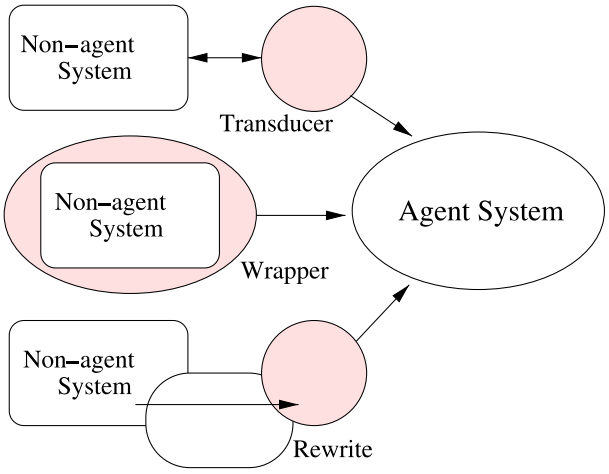

Fig. 6. Three approaches to agentification.

interface of the non-agent system needs to be known. On the other hand, to implement a Wrapper one needs availability to code. The existing software is modified by adding code to wrap the existing interface with a new one which can communicate in an ACL. Note here, that in system integration, when we use the term 'wrapper' we mean either the Transducer or the Wrapper approach. Finally, one could also Rewrite the non-agent system according to an agent paradigm. That amounts to a lot of programming work but one could potentially enhance the system's efficiency and capabilities.

Not surprisingly, FIPA defined an Agent Software Integration specification [80] which is concerned with how agents can connect to and make use of external software systems, that is systems that are external to and independent of an agents execution model—the Transducer approach to wrapping. However, nothing in the FIPA specifications prohibit anyone to actually implement an agent that makes use of the Wrapper approach; assuming access to code is provided, as long as it is otherwise a FIPA-compliant agent. In agent terminology, wrappers are usually called Resource Agents (RAs).

3.3. Adding data sources

In accordance with FIPA specifications all (non-agent) software systems (data sources) should be described by software descriptions[5] [80] which list the properties of the software system.

According to the 'Agent Software Integration Model' [80] FIPA supports another agent role: an agent that brokers a set of software descriptions to interested agents. New data sources can be added dynamically to the system just by providing a software description for the resource to the request broker. Subsequently, any

[5] Also known as *source descriptions* [81,82].

agents that require a service can query the request broker to get a list of the available agents.

3.4. Bioinformatics integration and agents

Throughout this paper we have discussed why agent technology is appropriate for the complex integration systems, particularly in bioinformatics. In summary:

- The layered approach of an ACL provides a flexible common medium to represent knowledge among agents.
- The ACL and the ontologies deal with the technical and semantic heterogeneities, respectively.
- RAs can wrap data sources. In addition using the FIPA 'Agent Software Integration' specification [80] new data sources can be added dynamically to the system.
- The adoption of agent (FIPA) standards help in making the first steps towards solving the sociological problem.
- The most popular multi-agent planning technique, task sharing, is almost identical to integration using mediation, and thus they could easily be combined.
- Adjustable Autonomy provides a potential solution to the problem of the confidence of the results in bioinformatics integration.
- Bioinformatics integration systems are complex distributed systems, which makes the use of agent technology a very good choice [40].

3.5. Agents, web services, and the grid

Fundamentally agent technology represents a software engineering paradigm. It provides a way to think of a system abstractly and enables designs were the basic concepts are expressed in terms of autonomous software entities, called agents. We can thus say that any software component exhibiting autonomous behaviour and able to communicate with its peers in a high-level semantically defined language *is* an agent. Of course there are so many parameters to be considered in the above statement—concerning efficiency, language expressivity, agent discovery, etc.—that significant standardisation efforts have been under way for many years now. After much research and testing, FIPA, the most prominent agent standardisation body, proposed the first standards in December 2002.

Meanwhile, another significant step towards the evolution of distributed component computing was made with the appearance of *Web Services* or more recently *Semantic* Web Services. Built on established technologies (HTML and XML) and on an unparallelled applications-, data-, and user-base, the Internet, Web Services have become wide-spread in a short period of time. This is driven by the need for acquiring computational services with similar ease to acquiring data from a web page. Web Services evolution can be related to agent technology. That becomes more clear when we examine the key concerns of Web Services: structured semantic data (XML, RDF, and XSD), service description (WSDL), communication protocol (SOAP), and a public service registry (UDDI). Finally, Web Services are also well supported by the standardisation efforts of the World Wide Web Consortium (W3C).

Because of these similarities many of the concepts from the two technologies have started to converge. Making Web Services autonomous and able to reason to accomplish their goals makes Web Services more like agents and rendering the latter widely available to the public, via the Internet, makes agents more like Web Services.

Similarly, Grid technology has a lot to gain from the high-level abstractions that the agent paradigm has to offer; use of agent protocols, negotiation, etc. For example, in myGrid, agents have been suggested to deal with personalisation issues such as acting on behalf of the user to automate system configuration and quality of service negotiation between a service publisher and a consumer [83].

Although we cannot predict with certainty the evolution/merging of these technologies, we believe that they can complement each other. All three technologies provide service-oriented functionality, which implies similarities (e.g., they all provide brokering facilities), but each one can contribute its more unique attributes:

- grid technology offers a large-scale distributed infrastructure,
- web technology provides formatting standards to represent data and knowledge, and
- agent technology offers autonomy and advanced communication between peers (e.g., protocols, negotiation, and personalisation), bridging the three technologies.

Consider an example, where software modules on a Grid exchange messages in an ACL, in which the content is expressed in RDF. One could imagine the integration potential made possible by combining these technologies. In many aspects, web services, grid, and agent technologies complement each other. However, currently only agent technology is self-sufficient enough to provide the advanced integration facilities mentioned in this paper on its own.

Agents are situated in a flexible and scalable distributed environment (Agent Platform [84]). They can represent data and knowledge in a variety of formatting standards including SL, KIF, XML, and RDF. Agents were designed with knowledge-sharing in mind and provide a common communication layer necessary for system integration. Moreover, they are good at addressing changes dynamically (e.g., new resources, different resource functionality, etc.), which is an important asset for an integration system applied in the

ever-changing biology domain. In addition, access to, interfacing with, and configuration of an agent system is very flexible; adjustable autonomy permits the user to decide when the system will act autonomously (automatically) or when according to user input. Furthermore, FIPA's standardisation efforts allow for the interoperability of agents developed from different organisations or vendors. That greatly increases the autonomy of the data sources, enabling their respective curators to retain total control of the resource's decisions, while at the same time offering a standardised public interface via an agent. A mutli-agent integration system demonstrating the above features is InfoSleuth [41].

4. Conclusions

Biology is a knowledge-intensive science and a large number of data sources are publicly available. Data sources are integrated to enable higher-level questions to be answered by combining their data. Integration is a complex task aiming to provide a unified view of the underlying resources, while eliminating potential technical and semantic heterogeneity. The mediation approach to integration is widely used and most bioinformatics integration systems make use—in different degree—of it.

Agent technology is a multi-disciplinary research field combining work from distributed systems, AI, and social (e.g., agent organisations and communication) and economic (e.g., auction protocols) sciences. Ever since its conception, its goal has been to develop techniques, methodologies, and software tools for knowledge sharing and reuse. Knowledge sharing is fundamental to integrating heterogeneous data sources, and as such, agent technology has much to offer to system integration. This becomes clearer after a detailed examination of agent properties and their usefulness for coping with bioinformatics integration challenges.

References

[1] Setubar J, Meidanis J. Introduction to computational biology. Brooks/Cole Publishing Company; 1997.

[2] Stein LP. Integrating biological databases. Nat Rev Gen 2003;4:337–45.

[3] Karp PD, Paley S, Zhu J. Database verification studies of SWISS-PROT and GenBank. Bioinformatics 2001;17(6):526–32.

[4] Attwood TK, Smith DJP. Introduction to bioinformatics. Addison Wesley Longman Education; 1999.

[5] Hasselbring W. Information system integration. Commun ACM 2000;43(6):33–8.

[6] Sheth A, Larson J. Federated database systems for managing distributed, heterogeneous, and autonomous databases. ACM Comput Surv 1990;22(3).

[7] Litwin W, Mark L, Roussopoulos N. Interoperability of multiple autonomous databases. ACM Comput Surv 1990;22(3): 267–93.

[8] El-Khatib HT, Williams MH, MacKinnon LM, Marwick DH. A framework and test-suite for assessing approaches to resolving heterogeneity in distributed databases. Inf Software Technol 2000;42:505–15.

[9] Sujansky W. Heterogeneous database integration in biomedicine. J Biomed Inform 2001;34:285–98.

[10] Hull R. Managing semantic heterogeneity in databases: a theoretical perspective. In: Proceedings of the Sixteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems. ACM Press; 1997. p. 51–61.

[11] Hakimpour F, Geppert, A. Resolving semantic heterogeneity in schema integration: an ontology based approach. In: Proceedings of the International Conference on Formal Ontology in Information Systems. ACM Press; 2001. p. 297–308.

[12] Harkey D, Orfali R. Client/server programming with Java and CORBA. New York: John Wiley; 1997.

[13] Davidson S, Overton C, Buneman P. Challenges in integrating biological data sources. J Comput Biol 1995;2(4).

[14] Gray PMD, Kemp GJL. Federated database technology for data integration—lessons from bioinformatics. In: Koslow SH, Huerta MF, editors. Electronic collaboration in science: progress in neuroinformatics, vol. 2. London: Lawrence Erlbaum Associates Inc; 2000. p. 45–72.

[15] Wiederhold G. Mediators in the architecture of future information systems. IEEE Comput 1992;21(3):38–50.

[16] Domenig R, Dittrich K. An overview and classification of mediated query systems. SIGMOD Rec 1999;28(3):63–72.

[17] Davidson SB, Overton C, Tannen V. BioKleisli: a digital library for biomedical researchers. Int J Digit Libr 1997;1(1).

[18] Buneman P, Libkin L, Suciu D, Tannen V, Wong L. Comprehension syntax. ACM SIGMOD Rec 1994;23(1):87–96.

[19] Goble C et al. Transparent access to multiple bioinformatics information sources. IBM Syst J 2001;40(2).

[20] Rector AL, Bechhofer S, Goble CA, Horrocks I, Nowlan WA, Solomon WD. The GALEN modelling language for medical terminology. AI Med 1996.

[21] Borgida A. Description logics in data management. IEEE Trans Knowl Data Eng 1995;7(5):671–82.

[22] K2. Available from: http://db.cis.upenn.edu/K2/index.html.

[23] Cattell RGG, Barry D, editors. The object database standard: ODMG 20. Morgan Kaufmann; 1997.

[24] Markowitz VM, Chen IA, Kosky AS, Szeto E. OPM: object-protocol model data management tools'97. In: Letovsky SI, editor. Bioinformatics databases and systems. Dordrecht: Kluwer Academic Publishers; 1999.

[25] Kemp GJL, Angelopoulos N, Gray PMD. Architecture of a mediator for a bioinformatics database federation. IEEE Trans Inf Technol Biomed 2002;6(2).

[26] Carter P, Coupaye T, Kreil DP, Etzold T. SRS: analyzing and using data from heterogenous textual databanks. In: Letovsky SI, editor. Bioinformatics databases and systems. Dordrecht: Kluwer Academic Publishers; 1999.

[27] Ritter O, Kocab P, Senger M, Wolf D, Suhai S. Prototype implementation of the integrated genomic database. Comput Biomed Res 1994;27(2):97–115.

[28] Haas LM, Schwarz PM, Kodali P, Kotlar E, Rice JE, Swope WC. DiscoveryLink: a system for integrated access to life sciences data sources. IBM Syst J (Deep computing for the life sciences) 2001;40(2).

[29] Carey M, Haas L, et al. Towards heterogeneous multimedia information systems: the garlic approach. In: Research issues in data engineering. IEEE Computer Society Press; 1995.

[30] Achard F, Vaysseix G, Barillot E. XML, bioinformatics and data integration. Bioinformatics 2001;17(2):115–25.

[31] Noy NF, McGuinness DL. Ontology Development 101: A Guide to Creating Your First Ontology. Technical Report KSL 01-05, Stanford University; 2001.
[32] Gruber T. Toward Pinciples for the Design of Ontologies Used for Knowledge Sharing. Technical Report KSL 93-04, Stanford University; 1993.
[33] Foster I, Kesselman C, Tuecke S. The anatomy of the grid: enabling scalable virtual organisations. Int J Supercomput Appl 2001;15(3).
[34] Foster I, Kesselman C, Nick J, Tuecke S. The physiology of the grid: an open grid services architecture for distributed systems integration 2002. Available from: http://www.globus.org/ogsa/.
[35] Wilkinson MD, Links M. BioMOBY: an open-source biological web services proposal. Brief Bioinform 2003;3(4):331–41.
[36] MyGRID. Available from http://www.mygrid.org.uk/.
[37] Baker PG et al. An ontology for bioinformatics applications. Bioinformatics 1999;15:510–20.
[38] Stevens R et al. Building a bioinformatics ontology using oil. IEEE Trans Inf Technol Biomed 2002;6(2):135–41.
[39] Ashburner M et al. Gene ontology: tool for the unification of biology. Nat Genet 2000;25:25–9.
[40] Jennings N. An agent-based approach for building complex software systems. Commun ACM 2001;44(4):35–41.
[41] Bayardo R, Bohrer W, Brice R, Cichocki A, Fowler J, Helal A, et al. Infosleuth: agent-based semantic integration of information in open and dynamic environments. In: ACM SIGMOD International Conference on Management of Data. ACL Press; 1997.
[42] RETSINA. Available from: www-2.cs.cmu.edu/~softagents/retsina_agent_arch.html.
[43] Garcia-Molina H, Papakonstantinou Y, Quass D, Rajaraman A, Sagiv Y, Ullman J, et al. The TSIMMIS approach to mediation: data models and languages. In: Next generation information technologies and systems. 1995.
[44] Bryson K, Luck M, Joy M, Jones D. Applying agents to bioinformatics in GeneWeaver. In: Cooperative information agents IV. Lecture notes in artificial intelligence, vol. 1860. 2000. p. 60–71.
[45] Bryson K, Luck M, Joy M, Jones D, Nicholas P, Bessieres P, et al. From geneweaver to agmial. In: Network tools and applications in biology—agents in bioinformatics. 2000.
[46] Decker K, Zheng X, Schmidt C. A multi-agent system for automated genomic annotation. In: Proceedings of the Fifth International Conference on Autonomous Agents. 2001.
[47] Mea VD. Agents acting and moving in healthcare scenario—a paradigm for telemedical collaboration. IEEE Trans Inf Technol Biomed 2001;5(1):10–3.
[48] Bryson K, Michael Luck, Mike Joy, Jones DT. Agent interaction for bioinformatics data management. Appl Artif Intell 2001;15(10):917–47.
[49] Salim Khan, Ravi Makkena, Foster McGeary, Keith Decker, William Gillis, Carl Schmidt, A multi-agent system for the quantitative simulation of biological networks. In: Proceedings of the Second International Joint Conference on Autonomous Agents and Multiagent Systems. ACM Press; 2003. p. 385–392.
[50] Karasavvas K, Burger A, Baldock RA. A multi-agent bioinformatics integration system with adjustable autonomy. In: PRICAI, Lecture Notes in Computer Science, vol. 2417. Springer; 2002. p. 492–501.
[51] Imai T, Matsuda H, Sekihara T, Nakanishi M, Hashimoto A. Implementing and integrated system for heterogeneous molecular biology databases with intelligent agents. In: Proceedings of the IEEE Pacific Rim Conference on Communications, Computers and Signal Processing. 1997. p. 807–810.
[52] Franklin S, Graesser A. Is it an agent, or just a program?: a taxonomy for autonomous agents. In: Third International Workshop on Agent Theories, Architectures, and Languages. Springer-Verlag; 1996.
[53] Nwana H. Software agents: an overview. Knowl Eng Rev 1996;11(3):205–44.
[54] Ringwald M, Eppig JT, Begley DA, Corradi JP, McCright IJ, Hayamizu TF, Hill DP, Kadin JA, Richardson JE. The mouse gene expression database (GXD). Nucleic Acids Res 2001;29:98–101.
[55] Davidson D, Bard J, Brune R, Burger A, Debreuil C, Hill W, Kaufman M, Quinn J, Stark M, Baldock R. The mouse atlas and graphical gene-expression database. Semin Cell Dev Biol 1997;8(5):509–17.
[56] Yule G. The study of language. Cambridge: Cambridge University Press; 1996.
[57] Austin J. How to do things with words. Cambridge, MA: Harvard University Press; 1962.
[58] Searle J. Speech acts: an essay in the philosophy of language. New York: Cambridge University Press; 1970.
[59] Foundation for Intelligent Physical Agents, Geneva, Switzerland. FIPA 97 Specification, Version 2.0, Part 2, Agent Communication Language. 1997.
[60] Finin T, Weber J, Wiederhold G, Genesereth M, Fritzson R, McGuire J, et al. Specification of the KQML agent communication language. Technical Report, DARPA knowledge sharing initiative, External Interfaces Working Group; 1993.
[61] FIPA SL Content Language Specification 2002. Availbale from: www.fipa.org/specs/fipa00008.
[62] Genesereth M, Fikes R. Knowledge Interchange Format, Version 3.0 Reference Manual. Technical Report 92-1. Computer Science Department. Stanford University; 1992.
[63] Sterling L, Shapiro E. The art of prolog. Cambridge, MA: MIT Press; 1986.
[64] Patil R, Fikes R, Patel-Schneider P, McKay D, Finin T, Gruber T, et al. TheDARPA knowledge sharing effort: progress report. In: Huhns M, Singh M, editors. Readings in agents. Morgan Kaufmann; 1997. p. 234–54 [chapter 3].
[65] Gruber T. Ontolingua: a mechanism to support portable ontologies. Technical Report KSL 91-66. Stanford University; 1992.
[66] Foundation for Intelligent and Physical Agents 1996. Available from: www.fipa.org.
[67] Weld D. An introduction to least commitment planning. AI Mag 1994;15(4):27–61.
[68] Weld D. Recent advances in AI planning. AI Mag 1999;20(2):93–123.
[69] Durfee E. Distributed problem solving and planning. In: Weiss G, editor. Multiagent systems: a modern approach to distributed artificial intelligence. Cambridge, MA: MIT Press; 2000. p. 121–64 [chapter 3].
[70] Ferber J. Multi-agent systems: an introduction to distributed artificial intelligence. Reading, MA: Addison-Wesley; 1999.
[71] Cesta A, D'Aloisi D, Collia M. Adjusting autonomy of agent systems. In: AAAI Spring Symposium on Agents with Adjustable Autonomy. 1999. p. 17–24.
[72] American Association for Artificial Intelligence. AAAI Spring Symposium on Agents with Adjustable Autonomy, March 1999.
[73] Scerri P, Pynadath DV, Tambe M. Why the elf acted autonomously: towards a theory of adjustable autonomy. In: Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems, 2002.
[74] Burstein M, Ferguson G, Allen J. Integrating agent-based mixed-initiative control with an existing multi-agent planning system. Computer Science Department 729, The University of Rochester, 2000.
[75] Parunak H. Industrial and practical applications of dai. In: Weiss G, editor. Multiagent systems: a modern approach to distributed artificial intelligence. Cambridge, MA: MIT Press; 2000. p. 377–421 [chapter 9].
[76] Booch G. Object-oriented analysis and design with applications. Reading, MA: Addison Wesley; 1994.

[77] Wooldridge M. Intelligent agents. In: Weiss G, editor. Multiagent systems: a modern approach to distributed artificial intelligence. Cambridge, MA: MIT Press; 2000. p. 27–77 [chapter 1].
[78] FIPA Ontology Service Specification 2001. Available from: www.fipa.org/specs/fipa00086.
[79] Genesereth MR, Ketchpel SP. Software agents. Commun ACM 1994;37(7):48–53.
[80] FIPA Agent Software Integration Specification 2001. Available from: www.fipa.org/specs/fipa00079.
[81] Levy A, Rajaraman A, Ordille J. Querying heterogeneous information sources using source descriptions. In: Twenty Second International Conference on Very Large Data Bases. Morgan Kaufmann; 1996. p. 251–262.
[82] Vassalos V, Papakonstantinou Y. Describing and using query capabilities of heterogeneous sources. In: Twenty Third International Conference on Very Large Data Bases. Morgan Kaufmann; 1997. p. 256–265.
[83] Moreau L, et al. On the use of agents in a bioinformatics grid. In: Proceedings of the Third International Symposium on Cluster Computing and the Grid. 2003.
[84] FIPA Agent Management Specification 2002. Available from: www.fipa.org/specs/fipa00023.