IMPACT: A Platform for Heterogenous Agents

Lecture Course given at Ushuaia, Argentina

October 2000

Jürgen Dix,
University of Koblenz and Technical University of Vienna

1. *IMPACT* Architecture
2. The Code Call Mechanism
3. Actions and Agent Programs
4. Regular Agents
5. Meta Agent Reasoning
6. Probabilistic Agent Reasoning
7. Temporal Agent Reasoning

Based on the book

Heterogenous Active Agents
(Subrahmanian, Bonatti, Dix,
Eiter, Kraus, Özcan and Ross),
MIT Press, May 2000.

Timetable:

- 10 minutes to explain what is going on. Some sentences for each chapter.
- Chapter 1 can be entirely done in the remaining time.

Prologue

1. MAS and DAI

2. Intelligent Agents

0.1 MAS and DAI

Three Important Questions

(Q1) What is an **Agent**?

(Q2) If some program *P* is not an agent, how can it be **transformed into an agent**?

(Q3) If (Q1) is clear, what kind of **Software Infrastructure** is needed for the interaction of agents? What services are necessary?

Definition 0.1 (Distributed Artificial Intelligence (DAI))

The area investigating systems, in which several autonomous acting entities work together to reach a given goal.

*The entities are called **Agents**, the area **Multiagent Systems**.*

Example: Robocup (simulation league, middle league)

Why do we need them?

Information systems are **distributed**, **open**, **heterogenous**.
We therefore need **intelligent, interactive agents**, that **act autonomously**.

Agent: Programs that are implemented on a platform and have sensors to react to the environment.

Intelligent: Performance measures, to reach goals. **Rational** vs. **omniscient**, **decision making**

Interactive: with other agents (or humans) by observing the environment. **Coordination:** Cooperation vs. Competition

MAS versus Classical DAI

(MAS) **Several Agents coordinate their knowledge and actions (semantics describes this).**

(DAI) **Particular problem is divided into smaller problems (nodes). These nodes have common knowledge. The solution method is given.**

Today DAI is often used synonymous with MAS: (1) as well as (2).

AI	**DAI**
Agent	**Multiple** Agents
Intelligence: Property of a **single** Agent	Intelligence: Property of **several** Agents
Cognitive Processes of a **single** Agent	**Social** Processes of **several** Agents

10 Desiderata

1. **Agents are for everyone!** We need a method to agentize given programs.
2. Take into account that **Data is stored in a wide variety of data structures, and data is manipulated by an existing corpus of algorithms.**
3. A theory of agents must *not* depend upon the set of actions that the agent performs. Rather, **the set of actions that the agent performs must be a *parameter* that is taken into account in the semantics.**

4. **Every agent should execute actions based on some *clearly articulated* decision policy.** A **declarative** framework for articulating decision policies of agents is imperative.
5. Any agent construction framework must allow agents to perform the following types of reasoning:
   - **Reasoning about its beliefs** about other agents.
   - **Reasoning about uncertainty** in its beliefs about the world and about its beliefs about other agents.
   - **Reasoning about time**.

   These capabilities should be viewed as *extensions* to a core agent action language.

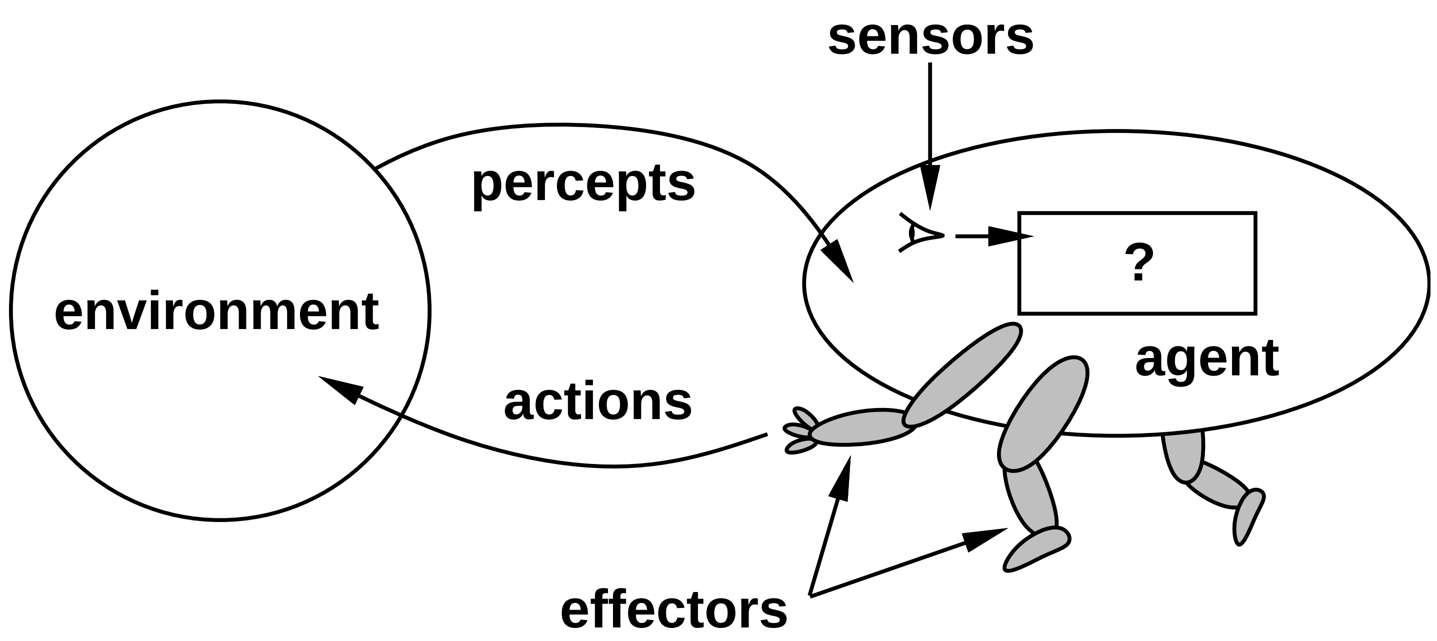
6. **Any infrastructure to support multiagent interactions *must* provide security.**
7. While the efficiency of the code underlying a software agent cannot be guaranteed (as it will vary from one application to another), **guarantees are needed that provide information on the performance of an agent relative to an oracle that supports calls to underlying software code.**

8. **We must identify efficiently computable *fragments* of the general hierarchy of languages alluded to above**, and our implementations must take advantage of the specific structure of such language fragments.
9. **A critical point is *reliability***—there is no point in a highly efficient implementation, if all agents deployed in the implementation come to a grinding halt when the agent “infrastructure” crashes.
10. The only way of testing the applicability of any theory is to **build a software system based on the theory**, to deploy a set of applications based on the theory, and to report on experiments based on those applications.

0.2 Intelligent Agents

Definition 0.2 (Agent)
An agent is a computer system that acts in its environment and executes autonomous actions to reach certain goals.

Learning, **Intelligence**. Environment is non-deterministic.



xbiff and **software demons** are agents. But certainly not intelligent.

Definition 0.3 (Intelligent Agents)
An intelligent agent is an agent with the following properties:

1. **Reactive**: *Reaction to changes in the environment at certain times to reach its goals.*
2. **Pro-active**: *Taking the initiative, goal-directed behaviour.*
3. **Social**: *Interaction with others to reach the goals.*

Pro-active alone is not sufficient (C-Programs): the environment can change during execution.

Difficulty: Right balance between pro-active and reactive!

Agents vs. Object Orientation

Objects have a

1. **state** (encapsulated): control over internal state,
2. message passing capabilities.

Java: private and public methods.

- Objects have control over their state, but **not over their behaviour**.
- An object can **not prevent others to use** its public methods.

Agents: They call other agents and request them to execute actions.

- Objects do it for free, agents do it for money.
- No analoga to **reactive**, **pro-active**, **social** in OO.
- **MAS are multi-threaded**: each agent has a control thread.
  In OO only the sytem as a whole posesses one.

Chapter 1. *IMPACT* Architecture

Overview

1.1 Three Szenarios

1.2 Agent Architecture

1.3 Server Architecture

1.4 Service Description Language

1 *IMPACT* Architecture

1.1 Three Szenarios

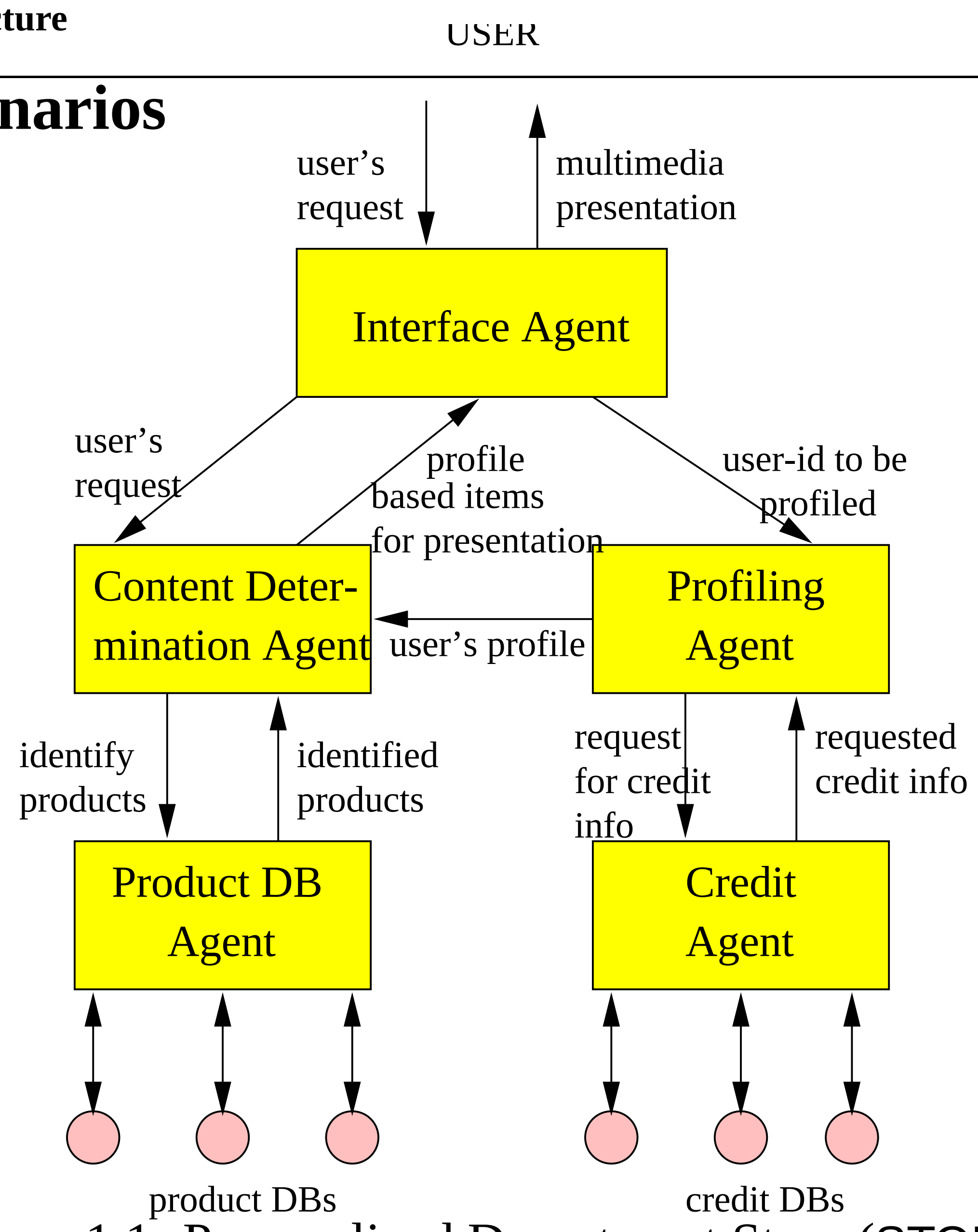


Figure 1.1: Personalized Department Store (STORE)

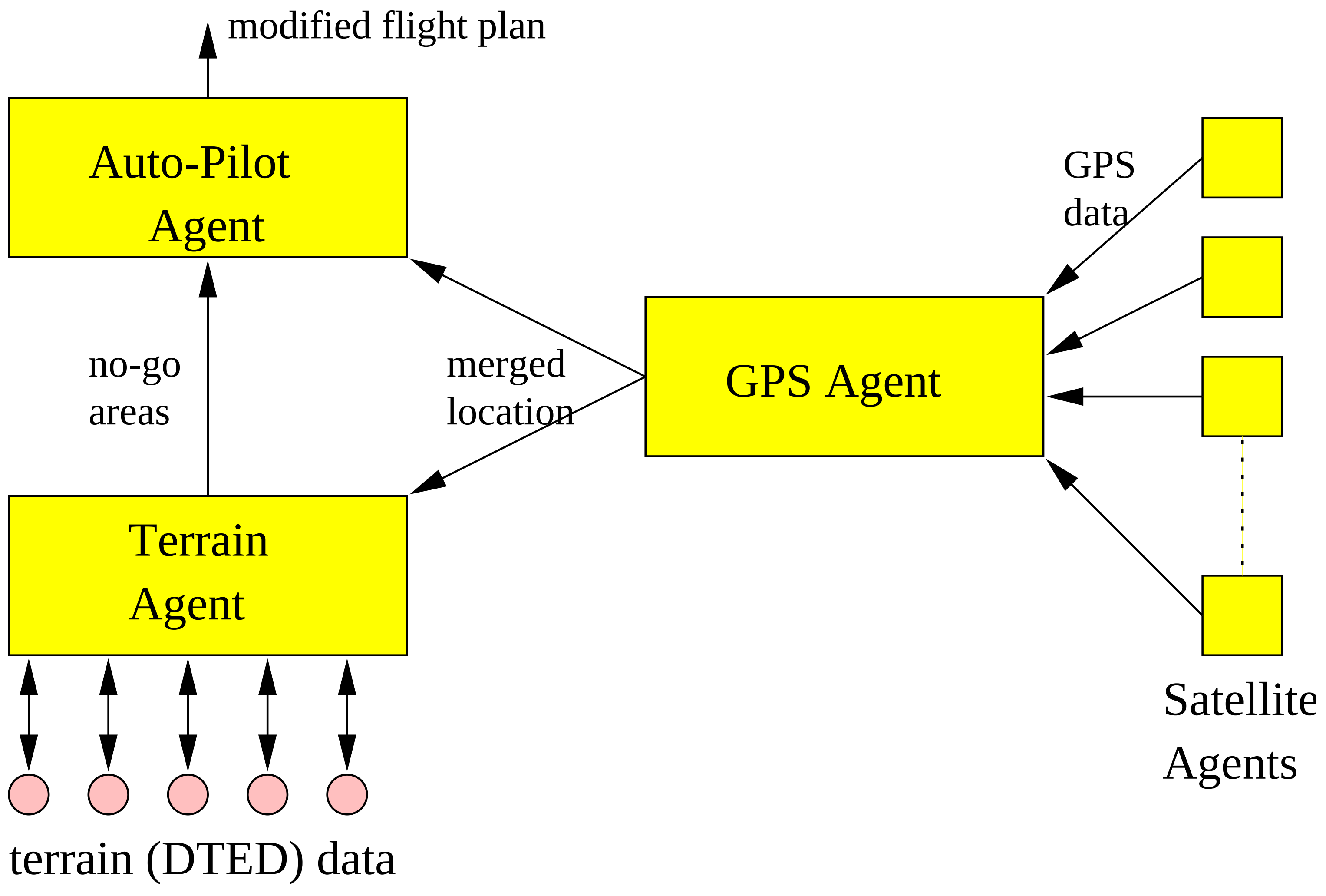


Figure 1.2: Interactions between Agents in CFIT Example

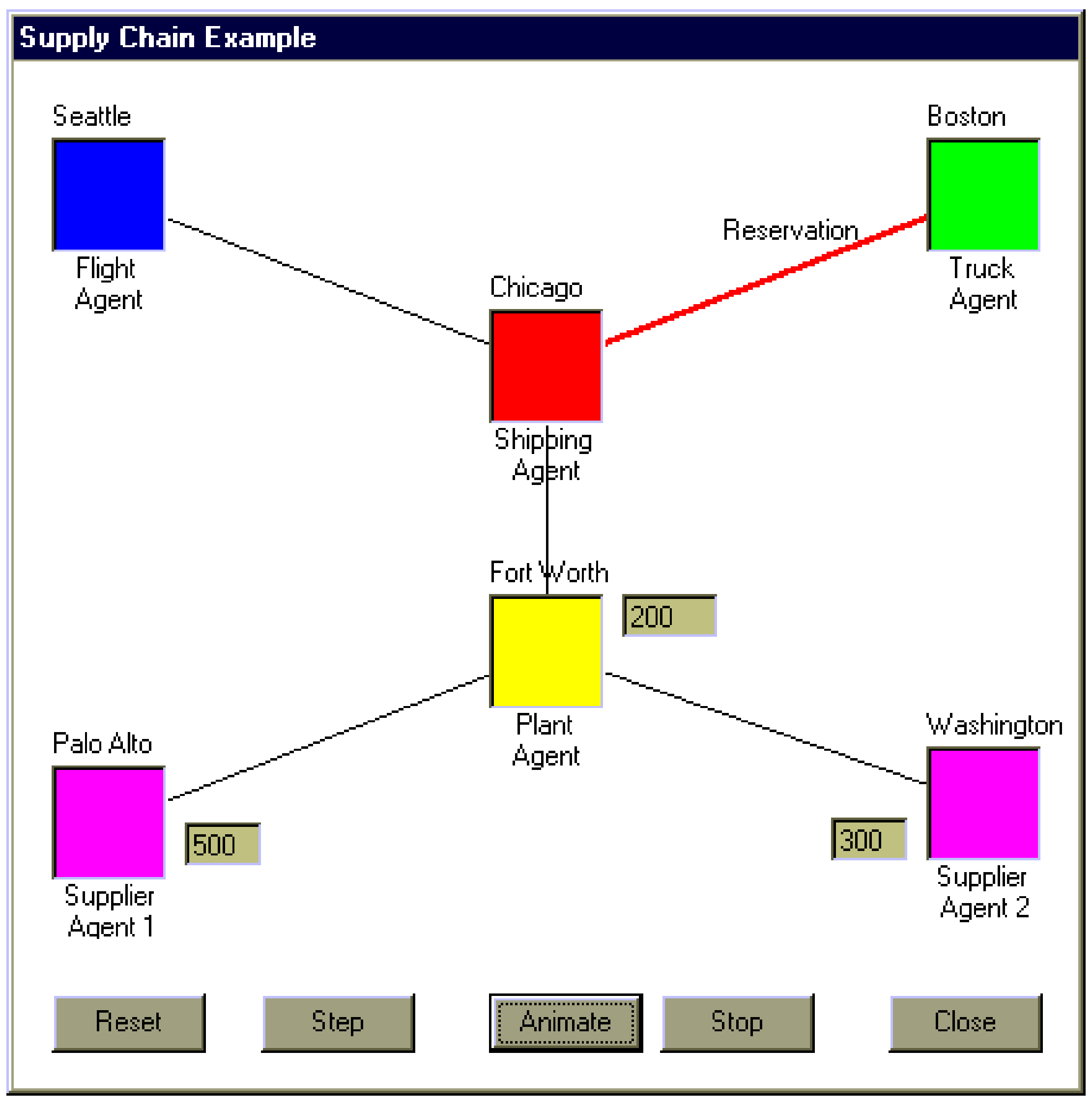


Figure 1.3: Agents in CHAIN Example

1.2 Agent Architecture

Four main categories:

1. In the first category, each agent has an associated **"transducer"** that converts all incoming messages and requests into a form that is intelligible to the agent. In general, in an n-agent system, we may need $O(n^2)$ transducers, which is clearly not desirable.

2. The second approach is based on **wrappers** which **"inject code into a program to allow it to communicate"** (Genesereth and Ketchpel 1994, p. 51). This idea is based on the principle that each agent has an associated body of code that is expressed in a common language used by other agents (or is expressed in one of a very small number of such languages).

3. The third approach described in (Genesereth and Ketchpel 1994) is to **completely rewrite the code** implementing an agent, which is obviously a very expensive alternative.

4. Last but not least, there is the **mediation approach** proposed by Wiederhold (1993), which assumes that all agents will communicate with a mediator which in turn may send messages to other agents. The mediation approach has been extensively studied (Arens, Chee, Hsu, and Knoblock 1993; Brink, Marcus, and Subrahmanian 1995; Chawathe, S., et al. 1994; Bayardo, R., et al. 1997).

   Here is the problem: Suppose all communications in the CFIT example had to go through such a mediator. Then if the mediator malfunctions or "goes down," the system as a whole is liable to collapse, leaving the plane in a precarious position.

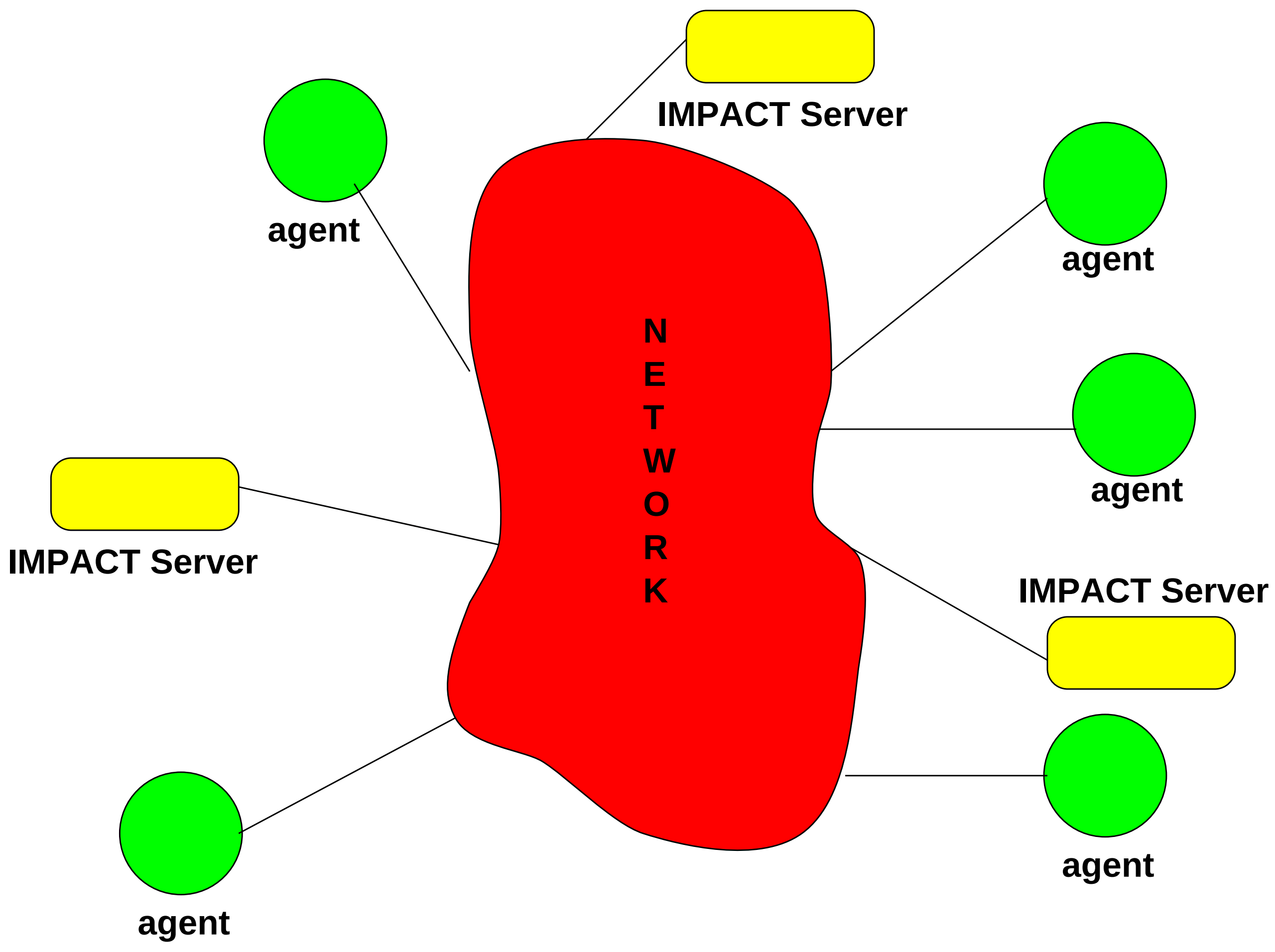


Figure 1.4: Overall *IMPACT* Architecture

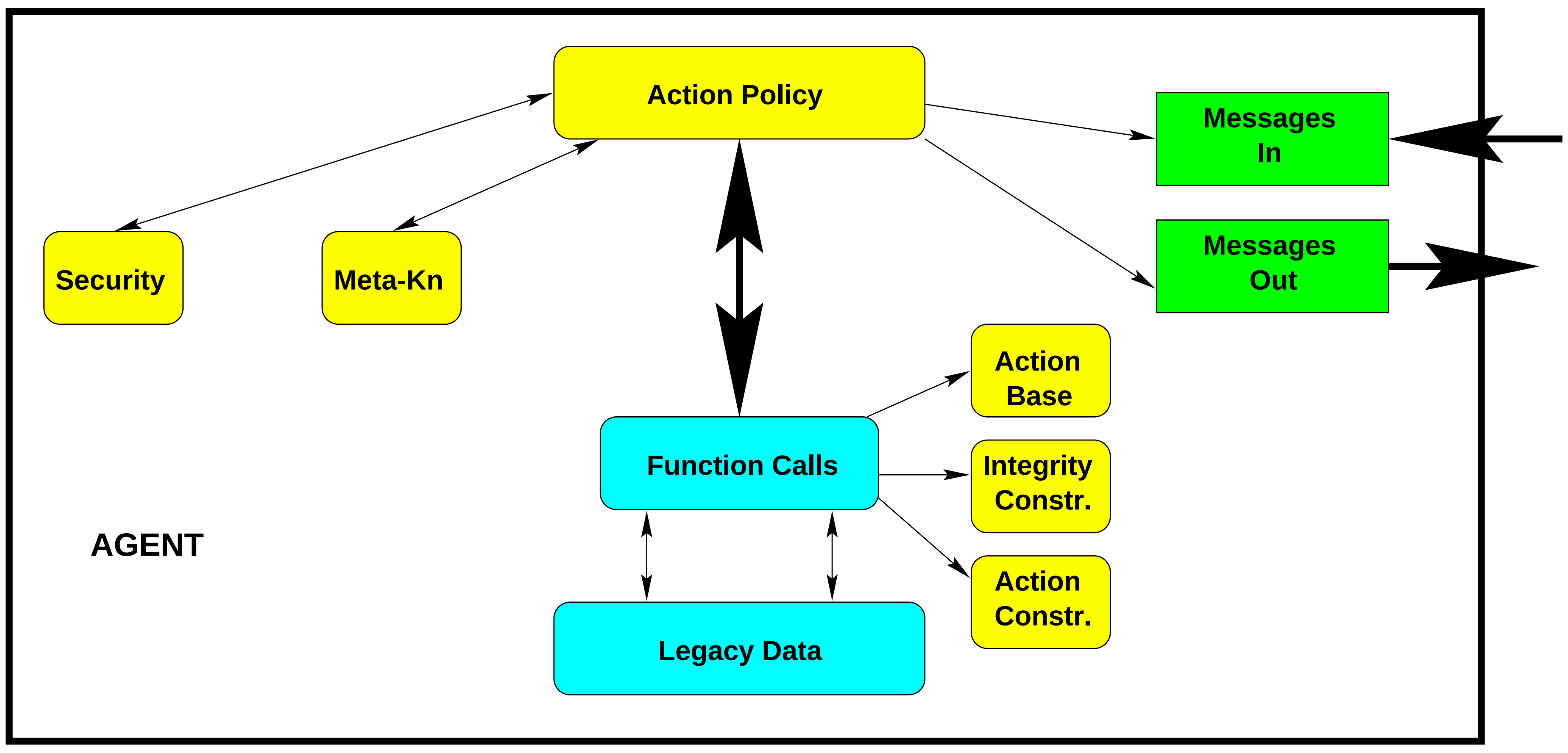


Figure 1.5: Basic Architecture of *IMPACT* Agents

1.3 Server Architecture

An *IMPACT* Server is actually a collection of the following servers:

Registration Server: This server is mainly used by the creator of an agent to specify the services provided by it and who may use those services.

Yellow Pages Server: This server processes requests from agents to identify other agents that provide a desired service.

Thesaurus Server: This server receives requests when new agent services are being registered as well as when the yellow pages server is searching for agents providing a service.

Type Server: This server maintains a set of class hierarchies containing information about different data types used by different agents, and the inclusion relationship(s) between them.

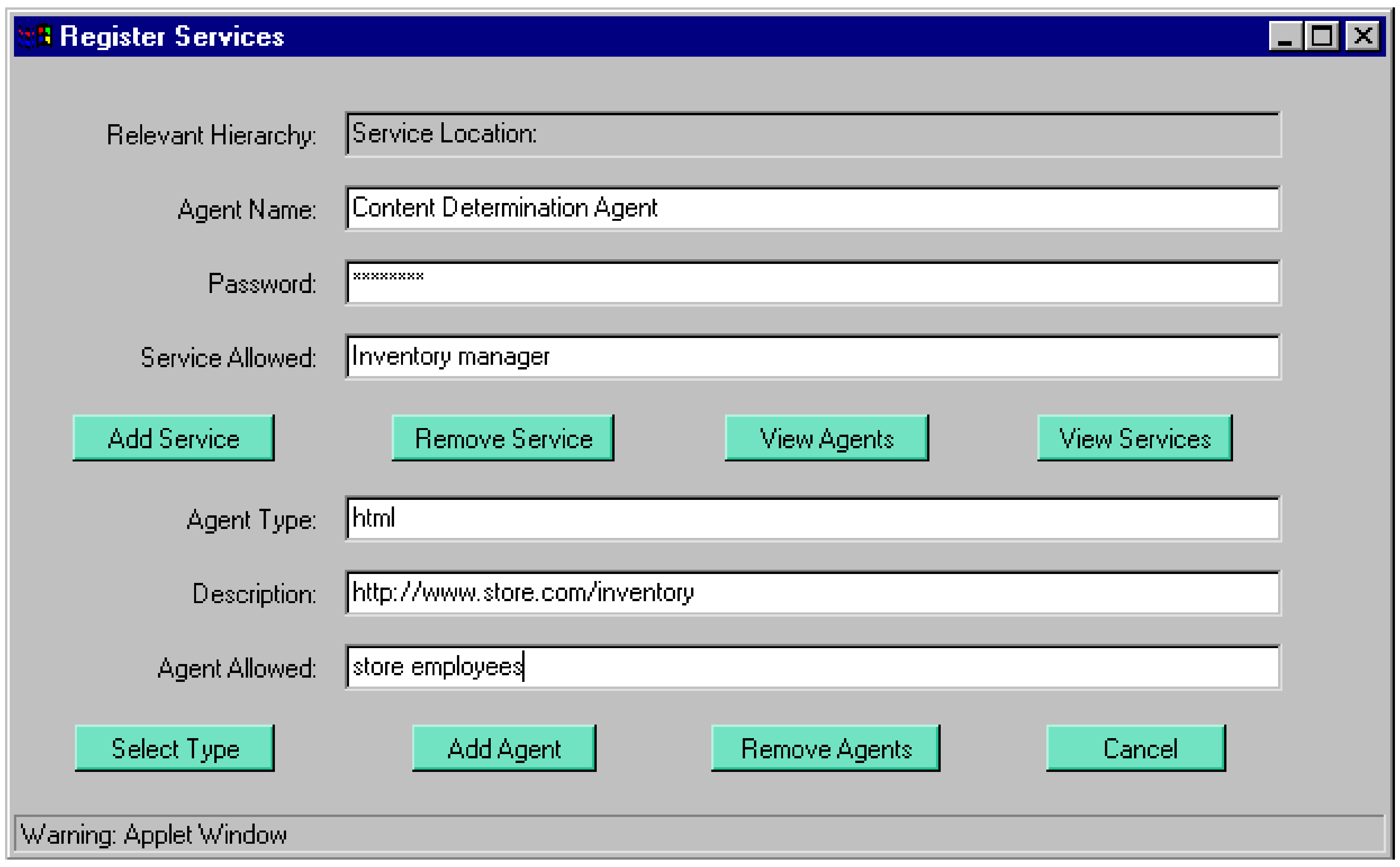


Figure 1.6: Agent/Service Registration Screen Dump

The user needs to specify the services of his agent. This is done in SDL (Service Description Language).

Definition 1.1 (Verbs, Nouns, nt(Nouns))
Suppose Verbs *is a set of verbs in English, and* Nouns *is a set of nouns in English.*

- *A* noun term *is either a noun or an expression of the form* $n_1(n_2)$ *where* n_1, n_2 *are both nouns.*
- $\mathsf{nt}(\mathsf{Nouns})$ *denotes the set of all syntactically valid noun terms generated by the set* Nouns.

Definition 1.2 (Service Name)
If $v \in \mathsf{Verbs}$ *and* $nt \in \mathsf{nt}$, *then* $v\!:\!nt$ *is called a* service name.

AGENT	SERVICES
credit	*provide* : *information(credit)* *provide* : *address*
profiling	*provide* : *user-profile* *classify* : *user*
productDB	*provide* : *description(product)* *identify* : *product*
contentDetermin	*prepare* : *presentation(product)* *determine* : *advertisement* *identify* : *items*
saleNotification	*identify* : *user-profile* *determine* : *items*

Table 1.1: Service List for the STORE example

AGENT	SERVICE
autoPilot	*maintain : course* *adjust : course* *return : control* *create : plan(flight)*
satellite	*broadcast : data(GPS)*
gps	*collect : data(GPS)* *merge : data(GPS)* *create : information(GPS)*
terrain	*generate : map(terrain)* *determine : area(no-go)*

Table 1.2: Service List for the CFIT example

AGENT	SERVICE
plant	*monitor* : *inventory*
	determine : *amount(part)*
	order : *part*
	notify : *supplier*
supplier	*monitor* : *available-stock*
	update : *stock*
	find : *airplane*
	prepare : *schedule(shipping)*
truck	*provide* : *schedule(truck)*
	manage : *freight*
	ship : *freight*

Table 1.3: Service List for the CHAIN example

What if one agent a seeks another one offering a service q_s?

We need to match q_s with other services in the yellow pages.

An agent looks for an agent offering the service *generate : map(ground)*.

Answer: CFIT terrain agent: *ground* and *terrain* are synonymous.

Suppose Σ is any set of English words, such that either all words in Σ are verbs, or all words in Σ are noun-terms. Furthermore, suppose $\sim$ is an arbitrary equivalence relation on Σ.

Definition 1.3 (Σ-node)
A Σ-node *is any subset* $N \subseteq \Sigma$ *that is closed under* $\sim$, *i.e.*

1. $x \in N \,\&\, y \in \Sigma \,\&\, y \sim x \Rightarrow y \in N$.
2. $x, y \in N \Rightarrow x \sim y$.

In other words, Σ-nodes *are equivalence classes of* Σ.

An agent looks for an agent offering the service *generate : map(area)*.

Answer: CFIT **terrain** agent: *area* can be specialized to *terrain*.

Definition 1.4 (Σ-Hierarchy)
A Σ-Hierarchy is a weighted, directed acyclic graph $\mathcal{SH} =_{def} (T, E, \wp)$ *such that:*

1. T *is set of nonempty Σ-nodes;*
2. *If* t_1 *and* t_2 *are different Σ-nodes in* T, *then* t_1 *and* t_2 *are disjoint;*
3. $\wp$ *is a mapping from* E *to* $\mathbb{Z}^+$ *indicating a positive distance between two neighboring vertices.*[a]

[a]We do not require $\wp$ to satisfy any metric axioms at this point in time.

Verb Hierarchy

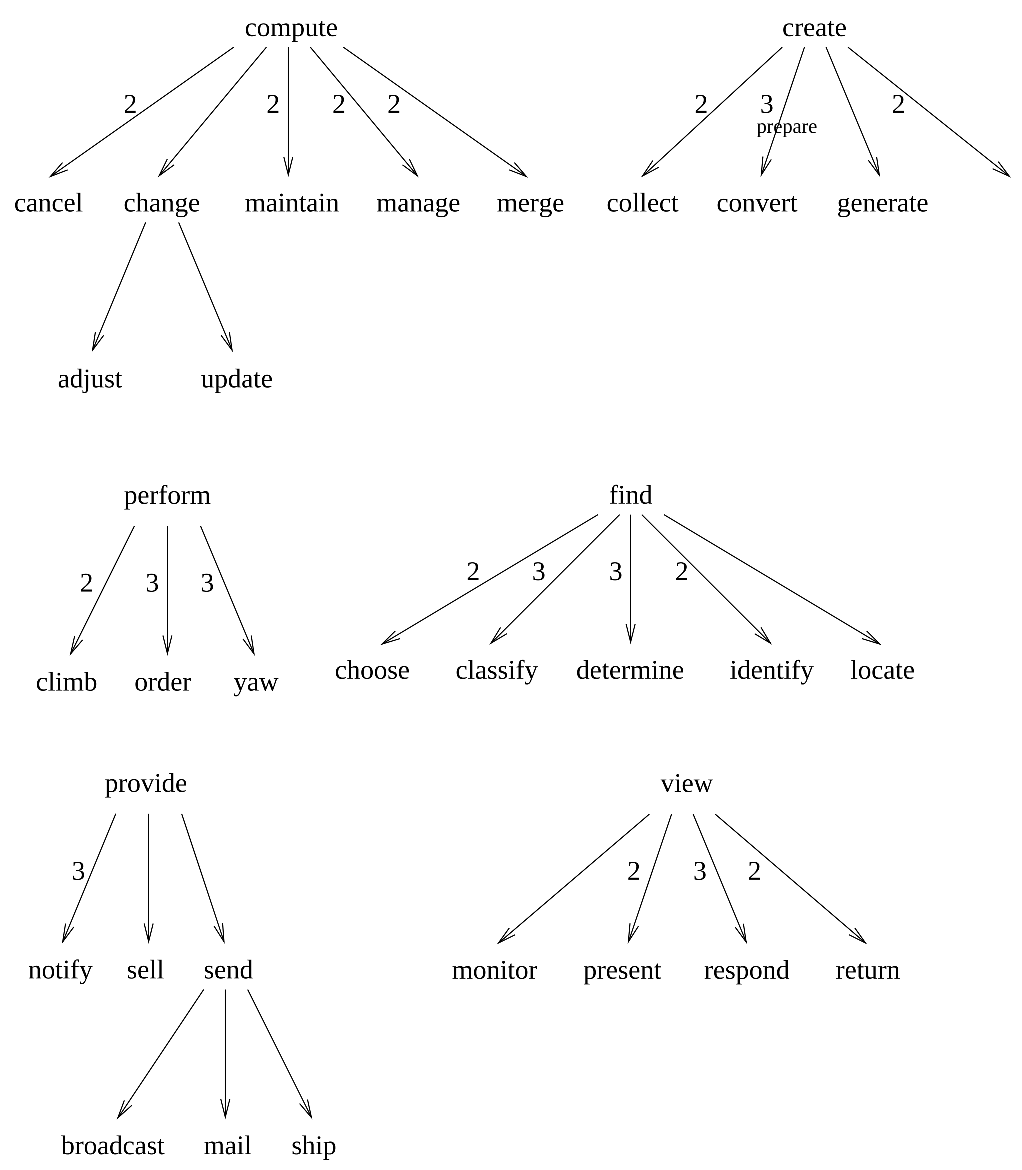


Figure 1.7: Verb Hierarchy (Missing Labels = 1)

Noun Hierarchy

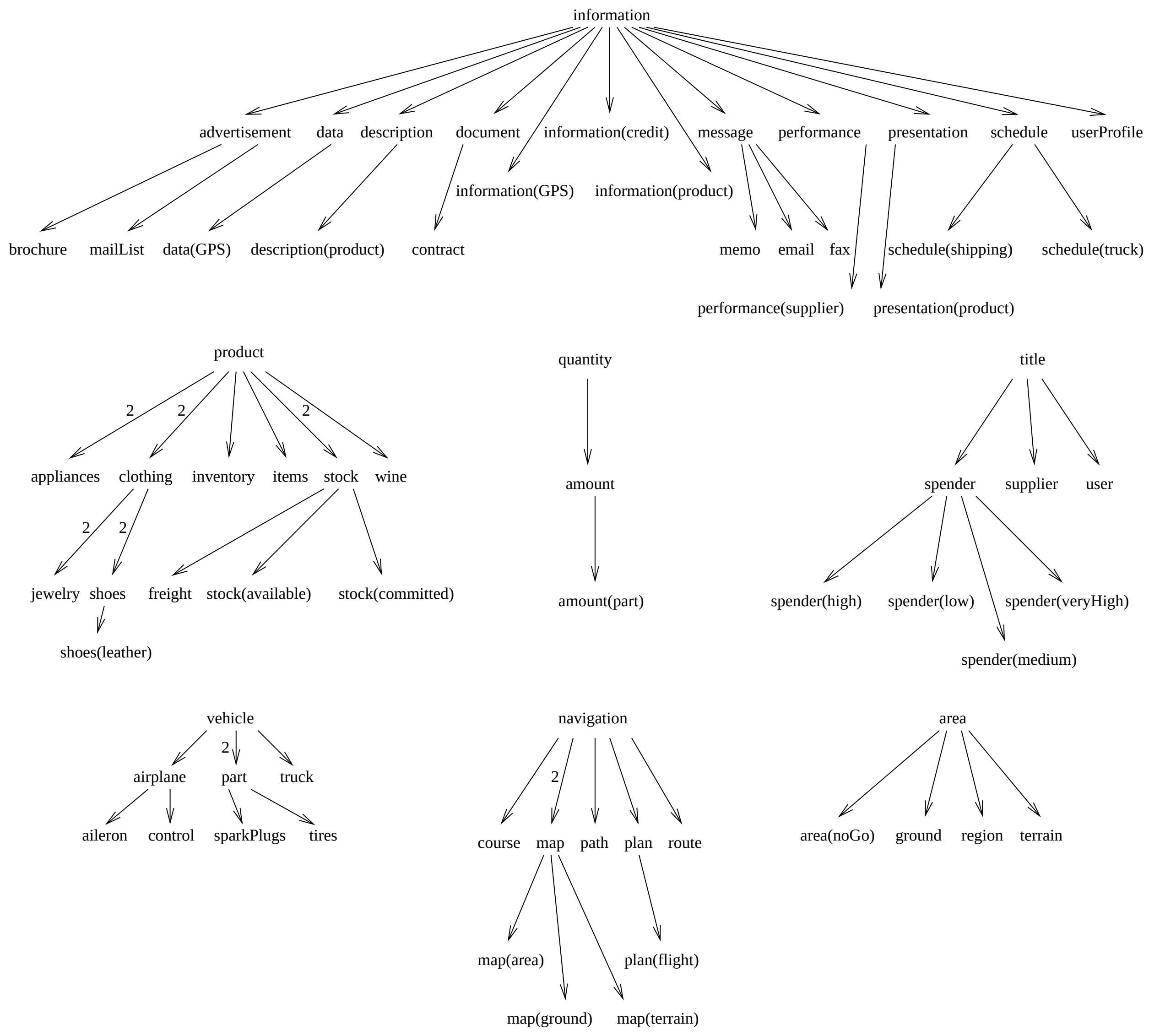


Figure 1.8: Noun-term Hierarchy

1.3.1 Distances

Definition 1.5 (Distance between two terms)

Given a Σ*-Hierarchy* $\mathcal{SH} =_{def} (T, E, \wp)$*, the distance between two terms,* $w_1, w_2 \in T$*, is defined as follows:*

$$\mathbf{d}_{\mathcal{SH}}(w_1, w_2) =_{def} \begin{cases} 0, & \text{if some } t \in T \text{ exists such that } w_1, w_2 \in t; \\ \mathbf{cost}(p_{min}), & \text{if there is an undirected path in } \mathcal{SH} \text{ between} \\ & w_1, w_2 \text{ and } p_{min} \text{ is the least cost such path;} \\ \infty, & \text{otherwise.} \end{cases}$$

It is easy to see that given any Σ-hierarchy, $\mathcal{SH} =_{def} (T, E, \wp)$, the distance function, $\mathbf{d}_{\mathcal{SH}}$ induced by it is well defined and satisfies the triangle inequality.

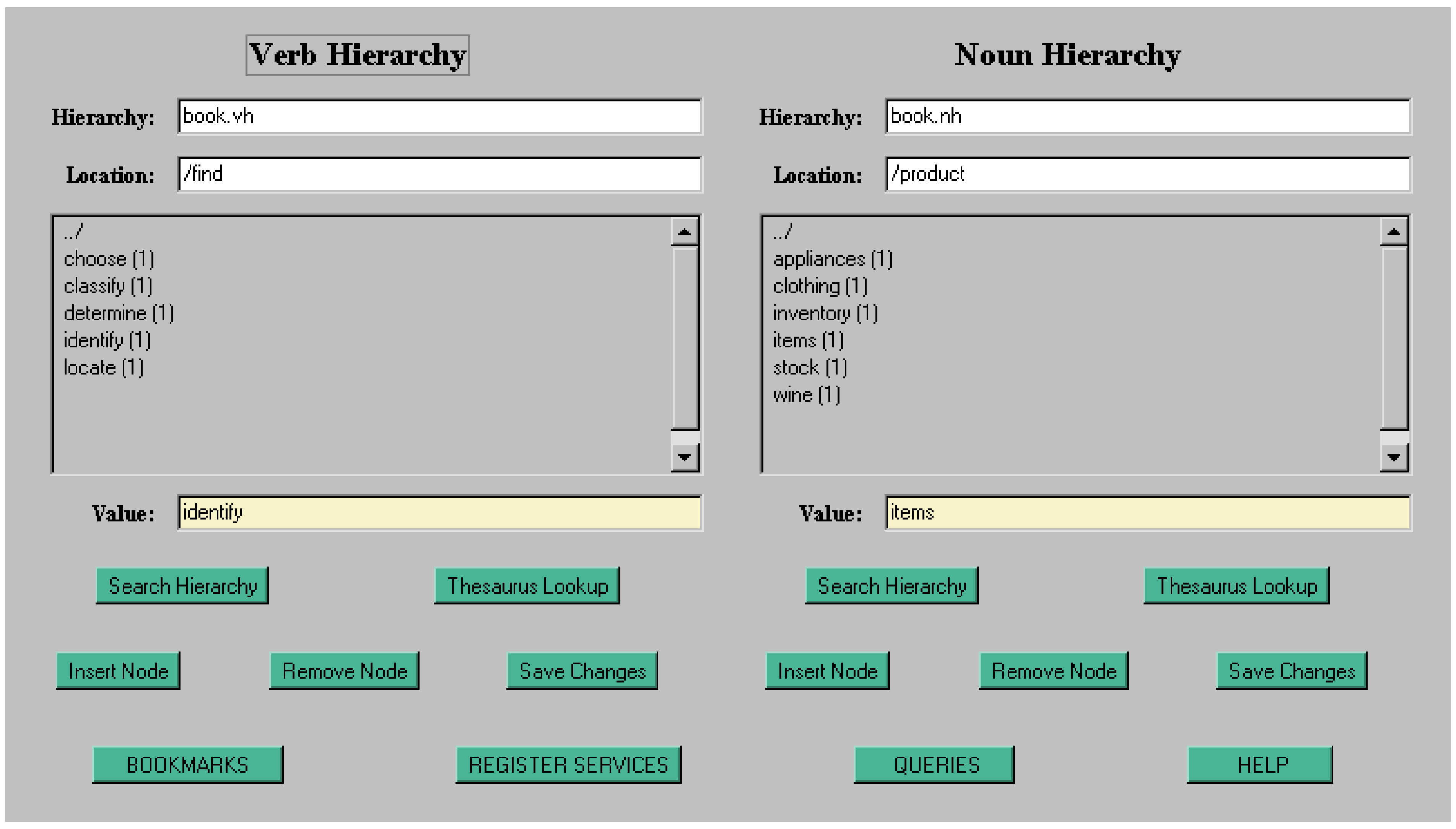


Figure 1.9: Hierarchy Browsing Screen Dump

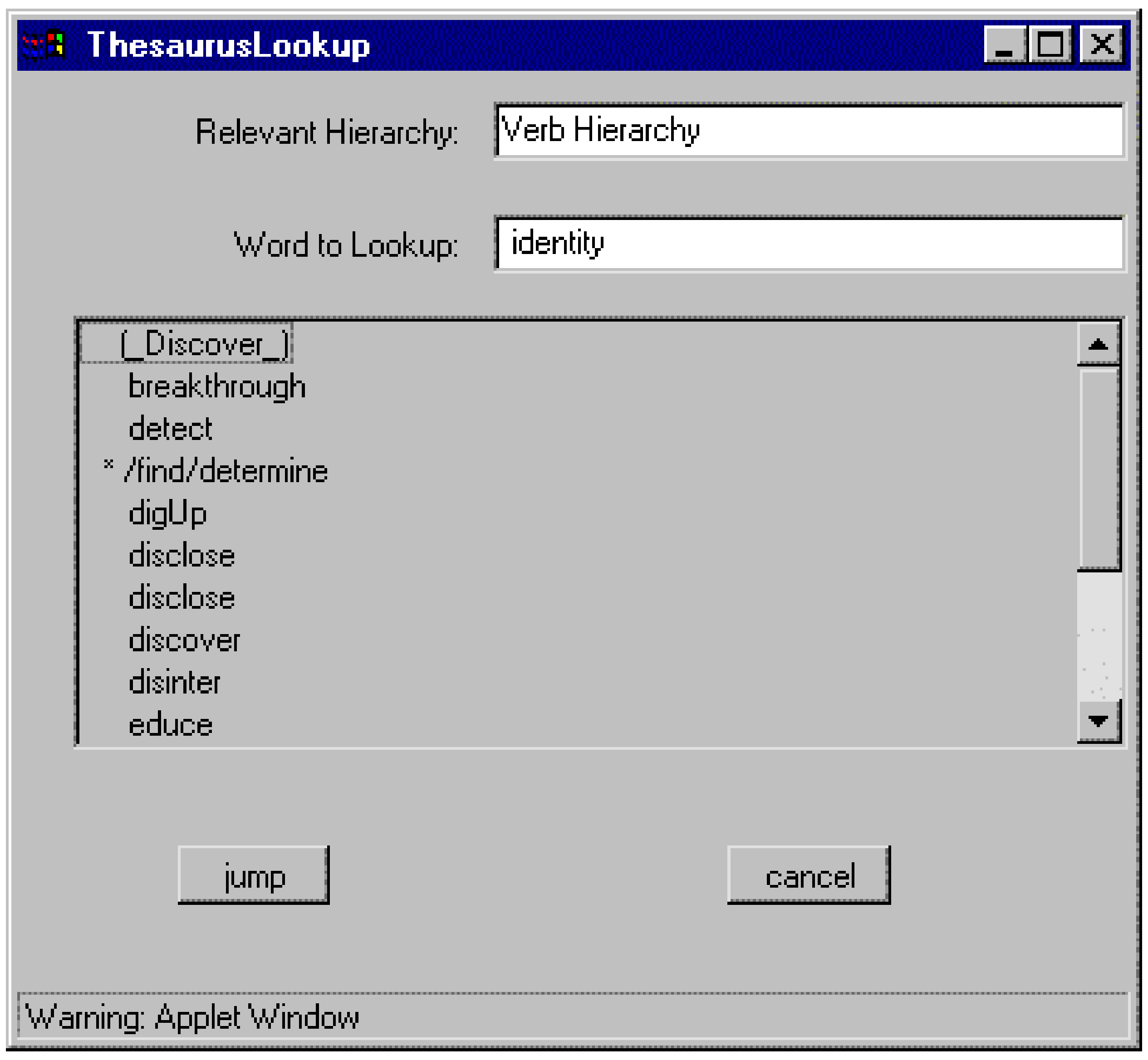


Figure 1.10: Thesaurus Screen Dump

1.4 Service Description Language

1.4.1 Definition of SDL

Service Name: This is a *verb : noun(noun)* expression describing the service.

Inputs: The user of a service will provide zero or more inputs. We also need a specification of what inputs are expected and which of these inputs are mandatory: “English” name for each input, and a semantic type for that input. For example: *Amount:* `Integer` specifies that we have an input called *Amount* of type `Integer` and *Part:* `PartName` specifies that we have an input called *Part* of type `PartName` (which could be an enumerated type).

Outputs: Each service must specify the outputs that it provides and each output is specified in the same way as an input.

Attributes: In addition, services may have attributes associated with them. Examples of such attributes include cost (for using the service), average response time for requests to that service, etc.

Definition 1.6 (Type/Type Hierarchy $(\mathcal{T}, \leq)$)
A type τ *is a set whose elements are called "values" of* τ*. The pair* $(\mathcal{T}, \leq)$ *is called a* type hierarchy *if* $\mathcal{T}$ *is a set of types and* $\leq$ *is a partial ordering on* $\mathcal{T}$*.*

Figure 1.11 provides a hierarchy associated with the three motivating examples.

Type Hierarchy

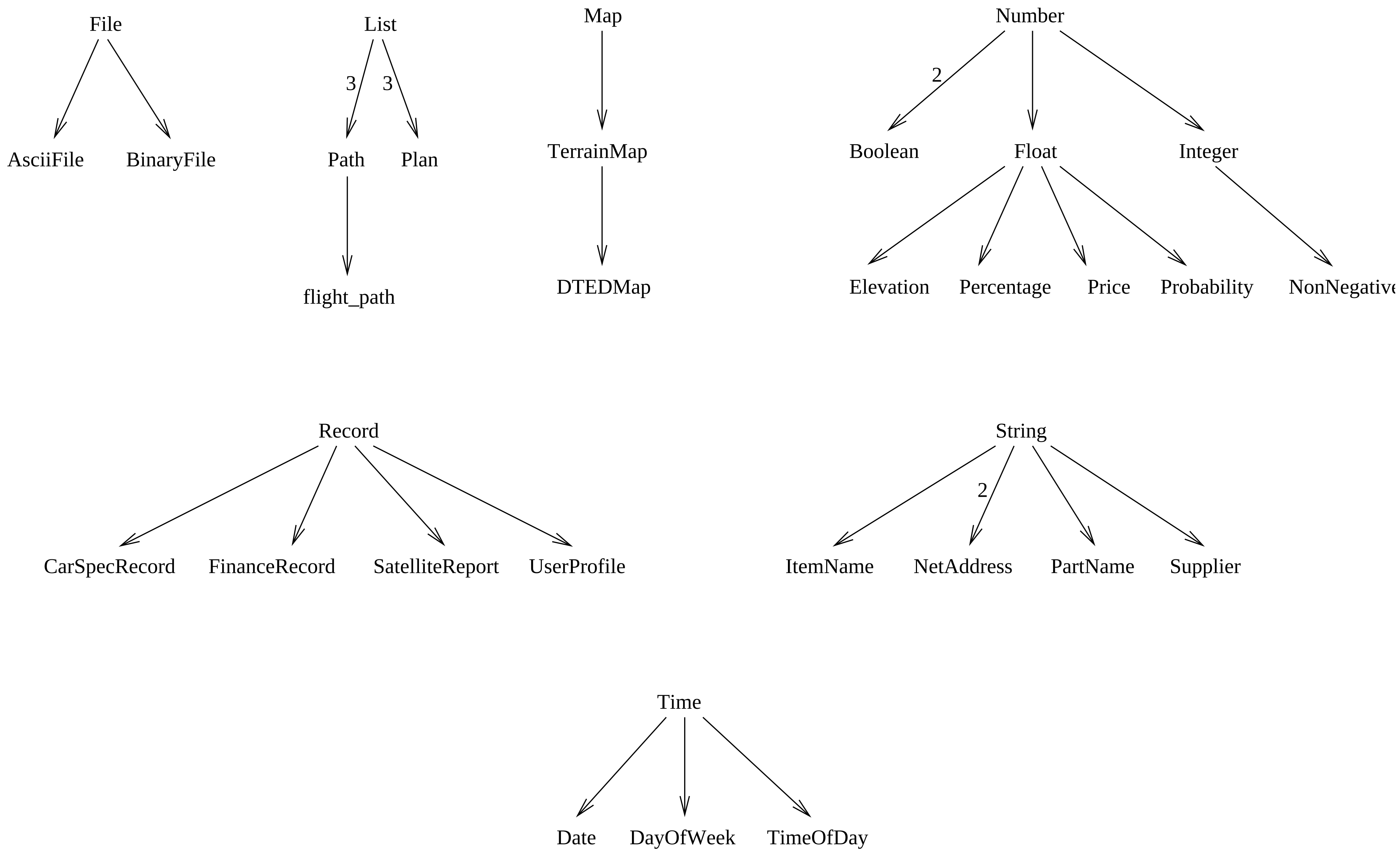


Figure 1.11: Example Type Hierarchy

Definition 1.7 (Set of Type Variables $V_{\mathcal{T}}$)
Associated with any type hierarchy $(\mathcal{T}, \leq)$, is a set $V_{\mathcal{T}}$ of symbols called type variables.

Intuitively, a type variable ranges over the values of a given type. For instance, `PartName` may be a type variable ranging over strings. When specifying the inputs required to invoke a service, we need to specify variables and their associated types. This is done in the usual way, as defined below.

Definition 1.8 (Items $s:\tau$)
If s is a variable ranging over objects of type τ, then $s:\tau$ is called an item.

> $s:\tau$ may be read as saying
> *"the variable s may assume values drawn from the type τ"*.

Definition 1.9 (Item Atom)
If $s\!:\tau$ *is an item, then* $\langle\mathtt{I}\rangle s\!:\tau\langle\backslash\mathtt{I}\rangle$ *(resp.* $\langle\mathtt{MI}\rangle s\!:\tau\langle\backslash\mathtt{MI}\rangle$ *) is called an* input *(resp.* mandatory input*) item atom, and* $\langle\mathtt{O}\rangle s\!:\tau\langle\backslash\mathtt{O}\rangle$ *is called an* output *item atom.*

Each input item is either ***mandatory*** or not. For example, $\langle\mathtt{MI}\rangle$*Location:* `String`$\langle\backslash\mathtt{MI}\rangle$ is a mandatory input item atom, while $\langle\mathtt{I}\rangle$*Nogo:* `TerrainMap`$\langle\backslash\mathtt{I}\rangle$ is a non-mandatory input item atom. The following are all valid output item atoms: $\langle\mathtt{O}\rangle$*Path1:* `Path`$\langle\backslash\mathtt{O}\rangle$, $\langle\mathtt{O}\rangle$*Specs:* `CarSpecRecord`$\langle\backslash\mathtt{O}\rangle$ and $\langle\mathtt{O}\rangle$*Financing_plan:* `FinanceRecord`$\langle\backslash\mathtt{O}\rangle$.

Definition 1.10 (Service Description)
Let sn be a service name, $i_1, \ldots, i_n$ be input item atoms, $mi_1, \ldots, mi_k$ be mandatory input item atoms, and $o_1, \ldots, o_r$ be output item atoms. Then,

$$
\begin{array}{ll}
\langle \mathtt{S} \rangle & sn \\
 & mi_1 \ldots mi_k \\
 & i_1 \ldots i_n \\
 & o_1 \ldots o_r \\
\langle \backslash \mathtt{S} \rangle &
\end{array}
$$

is called a service description.

Definition 1.11 (Item List)
If $s_1\!:\tau_1,\ldots,s_n\!:\tau_n$ *are* $n \geq 1$ *items, then* $\langle\mathtt{I}\rangle s_1\!:\tau_1,\ldots,s_n\!:\tau_n\langle\backslash\mathtt{I}\rangle$ *is an* input item list, *which is a shorthand for* $\langle\mathtt{I}\rangle s_1\!:\tau_1\langle\backslash\mathtt{I}\rangle\cdots\langle\mathtt{I}\rangle s_1\!:\tau_n\langle\backslash\mathtt{I}\rangle$*; also,* $\langle\mathtt{MI}\rangle s_1\!:\tau_1,\ldots,s_n\!:\tau_n\langle\backslash\mathtt{MI}\rangle$ *and* $\langle\mathtt{O}\rangle s_1\!:\tau_1,\ldots,s_n\!:\tau_n\langle\backslash\mathtt{O}\rangle$ *are* mandatory input item lists *and* output item lists, *respectively, which are shorthands for the items* $\langle\mathtt{MI}\rangle s_1\!:\tau_1\langle\backslash\mathtt{MI}\rangle\cdots\langle\mathtt{MI}\rangle s_1\!:\tau_n\langle\backslash\mathtt{MI}\rangle$ *and* $\langle\mathtt{O}\rangle s_1\!:\tau_1\langle\backslash\mathtt{O}\rangle\cdots\langle\mathtt{O}\rangle s_1\!:\tau_n\langle\backslash\mathtt{O}\rangle$*, respectively.*

$\langle \mathtt{S} \rangle$ *classify: user*

$\langle \mathtt{MI} \rangle$*ssn:* `String`$\langle \backslash \mathtt{MI} \rangle$

$\langle \mathtt{I} \rangle$*name:* `String`$\langle \backslash \mathtt{I} \rangle$

$\langle \mathtt{O} \rangle$*class:* `UserProfile`$\langle \backslash \mathtt{O} \rangle$

$\langle \backslash \mathtt{S} \rangle$

This service may take as input, the name and the social security number of a user, and provide as output, a classification of the user as a "low," "medium," "high," or "very high" spender. The social security number is a mandatory input, whereas the **name** is **optional** as it can be uniquely determined from a person's social security number.

$\langle \mathtt{S} \rangle$ *create: plan(flight)*

$\langle \mathtt{MI} \rangle$ *Location:* $\mathtt{SatelliteReport}$, *Flightroute:* $\mathtt{Path}$, *Nogo:* $\mathtt{Map} \langle \backslash \mathtt{MI} \rangle$

$\langle \mathtt{O} \rangle$ *Plan:* $\mathtt{Plan} \langle \backslash \mathtt{O} \rangle$

$\langle \backslash \mathtt{S} \rangle$

This service takes three mandatory inputs (the location of the plane, the allocated flight route of the plane, and a set of Nogo areas), and generates a modified flight path for the plane.

$\langle\texttt{S}\rangle$ *monitor: availablestock*

$\langle\texttt{MI}\rangle$*Amount:* $\texttt{Integer}$, *Partid:* $\texttt{String}\langle\backslash\texttt{MI}\rangle$

$\langle\texttt{I}\rangle$*Name:* $\texttt{String}\langle\backslash\texttt{I}\rangle$

$\langle\texttt{O}\rangle$*Status:* $\texttt{String}\langle\backslash\texttt{O}\rangle$

$\langle\backslash\texttt{S}\rangle$

This service takes the *Amount* and *Part_id* of the requested part as mandatory inputs, and the *name* of the requested part as an optional input. The *Name* of a *part* maybe determined from its *Part_id*. This service returns as output the string `amount_available` or `amount_not_available`.

1.4.2 Metric and Matchmaking

Up to now, we defined distances between verbs and between noun-terms. But we need to have a **distance between service-names**!

Definition 1.12 (Composite Distance Function cd)
Suppose we have two different sets of words Σ_1 *and* Σ_2 *with* Σ_1*-hierarchy* $\mathcal{SH}_1 =_{def} (T_1, E_1, \wp_1)$ *and* Σ_2*-hierarchy* $\mathcal{SH}_2 =_{def} (T_2, E_2, \wp_2)$. *Let* $\mathbf{d}_1, \mathbf{d}_2$ *be the distance functions induced by* $\mathcal{SH}_1, \mathcal{SH}_2$*, respectively. Consider two pairs of words,* $\langle w_1, w_1' \rangle, \langle w_2, w_2' \rangle \in \Sigma_1 \times \Sigma_2$. *A* composite distance *function* **cd** *is any mapping from* $(\Sigma_1 \times \Sigma_2) \times (\Sigma_1 \times \Sigma_2)$ *to* $\mathbb{Z}^+$ *such that:*

1. $\mathbf{cd}(\langle w_1, w_1' \rangle, \langle w_2, w_2' \rangle) = \mathbf{cd}(\langle w_2, w_2' \rangle, \langle w_1, w_1' \rangle)$ *(Symmetry)*

2. $\mathbf{cd}(\langle w_1, w_1' \rangle, \langle w_1, w_1' \rangle) = 0$ *(Ipso-distance)*

3. *If* $\mathbf{d}_1(w_1, w_2) \leq \mathbf{d}_1(w_1, w_3)$, *then*
   $\mathbf{cd}(\langle w_1, w_1' \rangle, \langle w_2, w_2' \rangle) \leq \mathbf{cd}(\langle w_1, w_1' \rangle, \langle w_3, w_2' \rangle)$ *(Expansion of* $\mathbf{d}_1$*)*

4. *If* $\mathbf{d}_2(w_1', w_2') \leq \mathbf{d}_2(w_1', w_3')$, *then*
   $\mathbf{cd}(\langle w_1, w_1' \rangle, \langle w_2, w_2' \rangle) \leq \mathbf{cd}(\langle w_1, w_1' \rangle, \langle w_2, w_3' \rangle)$ *(Expansion of* $\mathbf{d}_2$*)*

5. $\mathbf{cd}(\langle w_1, w_1' \rangle, \langle w_3, w_3' \rangle) \leq \mathbf{cd}(\langle w_1, w_1' \rangle, \langle w_2, w_2' \rangle) +$
   $\mathbf{cd}(\langle w_2, w_2' \rangle, \langle w_3, w_3' \rangle)$ *(Triangle Inequality).*

Example 1.1 (Composite Distances)

Let $\mathbf{d}_1$ *and* $\mathbf{d}_2$ *be distances defined as in Section 1.2 on the verb and noun-term hierarchies given in Figure 20 on page 20 and Figure 21 on page 21, respectively. Moreover, let the composite distance function be defined as*

$$\mathbf{cd}(\langle w_1, w_1' \rangle, \langle w_2, w_2' \rangle) =_{def} \mathbf{d}_1(w_1, w_2) + \mathbf{d}_2(w_1', w_2').$$

Now consider the following two pairs: $\langle \mathit{provide}, \mathit{information} \rangle$ *and* $\langle \mathit{broadcast}, \mathit{data(GPS)} \rangle$*. As can be seen from Figure 20 on page 20, the distance between provide and broadcast is*

$$\mathbf{d}_1(\mathit{provide}, \mathit{broadcast}) = 2,$$

as is the distance between information and data(GPS) (see Figure 21 on page 21).

Thus, the composite distance between these two pairs is given by

$$\begin{aligned}
&\mathbf{cd}(\langle \mathit{provide}, \mathit{information}\rangle, \\
&\quad \langle \mathit{broadcast}, \mathit{data(GPS)}\rangle) &= \mathbf{d}_1(\mathit{provide}, \mathit{broadcast}) + \mathbf{d}_2(\mathit{information}, \mathit{data(GPS)}) \\
& &= 4.
\end{aligned}$$

As another example, consider the pairs $\langle \mathit{identify}, \mathit{items}\rangle$ and $\langle \mathit{determine}, \mathit{product}\rangle$. In this case, as given by Figure 20 on page 20, the distance $\mathbf{d}_1(\mathit{identify}, \mathit{determine})$ between *identify* and *determine* is 5. And from Figure 21 on page 21, the distance between *items* and *product* is $\mathbf{d}_2(\mathit{items}, \mathit{product}) = 1$. Then, the composite distance between $\langle \mathit{identify}, \mathit{items}\rangle$ and $\langle \mathit{determine}, \mathit{product}\rangle$ will be the sum of their verb and nounterm distances, i.e., 6.

What if we are looking for the distance between $n_1(n'_1)$ and $n_2(n'_2)$, but these terms do not occur (only n_1, n'_1, n_2, n'_2 are in nt)?

Then we use **cd** where $\Sigma_1 := \Sigma_2 := \mathsf{nt}$.

What if we are looking for the distance between $n_1(n_2)$ and n_1 but the term $n_1(n_2)$ does not ocurr? (There might be a synonym n_3 for n_2 s.t. $n_1(n_3)$ ocurrs.)

Then we roughly estimate: see Definition of $\mathbf{d_G}$.

Definition 1.13 (The Function $\mathbf{d_G}$)
We interpret, n_1 as $n_1(general)$, e.g., information as information(general), and assume that a function denoted by $\mathbf{d_G}$ for computing the distance between any noun n and general is given to the system. E.g.: ($w_1, \ldots, w_k$ are the weights of all edges between the Noun-Term-node and any of its neighboring vertices)

$$\mathbf{d_G}(n, general) =_{def} max(w_1, \ldots, w_k).$$

Example 1.2 (Distances)

When n is the noun-term map or navigation, $\mathbf{d_G}(n, general) = 2$ but when n is plan or route, $\mathbf{d_G}(n, general) = 1$.

Consider a query which asks for map(region). Which noun-term should we consider first?

Although there is no noun-term in our hierarchy named map(region), there are noun-terms for both map and region. Recall that $\mathbf{d_G}(map, general) = 2$. If we can find a noun-term n with a distance of 2 or less from map(region), we should start at n. Otherwise, we should start at map.

In our current example, we should start at map(area) as region has a distance of 1 from area and so map(area) has a distance of $1 < 2$. However, if we were looking for map(city), there is no noun-term with a distance of 2 or less so we should start at map.

Matchmaking

It is easy to define **find_nn**: An algorithm to solve the k-nearest neighbor problem.

Given a pair $\langle v, nt \rangle$ specifying a desired service, this algorithm will return a set of k agents that provide the most closely matching services.

Closeness between $\langle v, nt \rangle$ and another pair $\langle v', nt' \rangle$ is determined by using

1. the distance functions associated with the verb and noun-term hierarchies,
2. a composite distance function **cd** specified by the agent invoking the **find_nn** algorithm.

Range Computations

The **range** Algorithm answers queries of the form

"Find all agents that provide a service $vnt = \langle V', NT' \rangle$ which is within a distance D of a requested service $vnt = \langle V, NT \rangle$".

1.4.3 Simulation Results

We are interested in

- the efficiency of finding *similar* services and
- the quality of the *matching* services provided as the output.

Performance Results

Based on a *NASA* hierarchy consisting of 17,445 words (for experimental purposes, the same hierarchy was used as both a verb and a noun hierarchy, although the *IMPACT* prototype uses different hierarchies). Weights on all edges in the hierarchies were assumed to be 1 and the composite distance function was taken to be `sum`.

The algorithms were implemented in C++ and the experiments were conducted on a Sun Sparc.

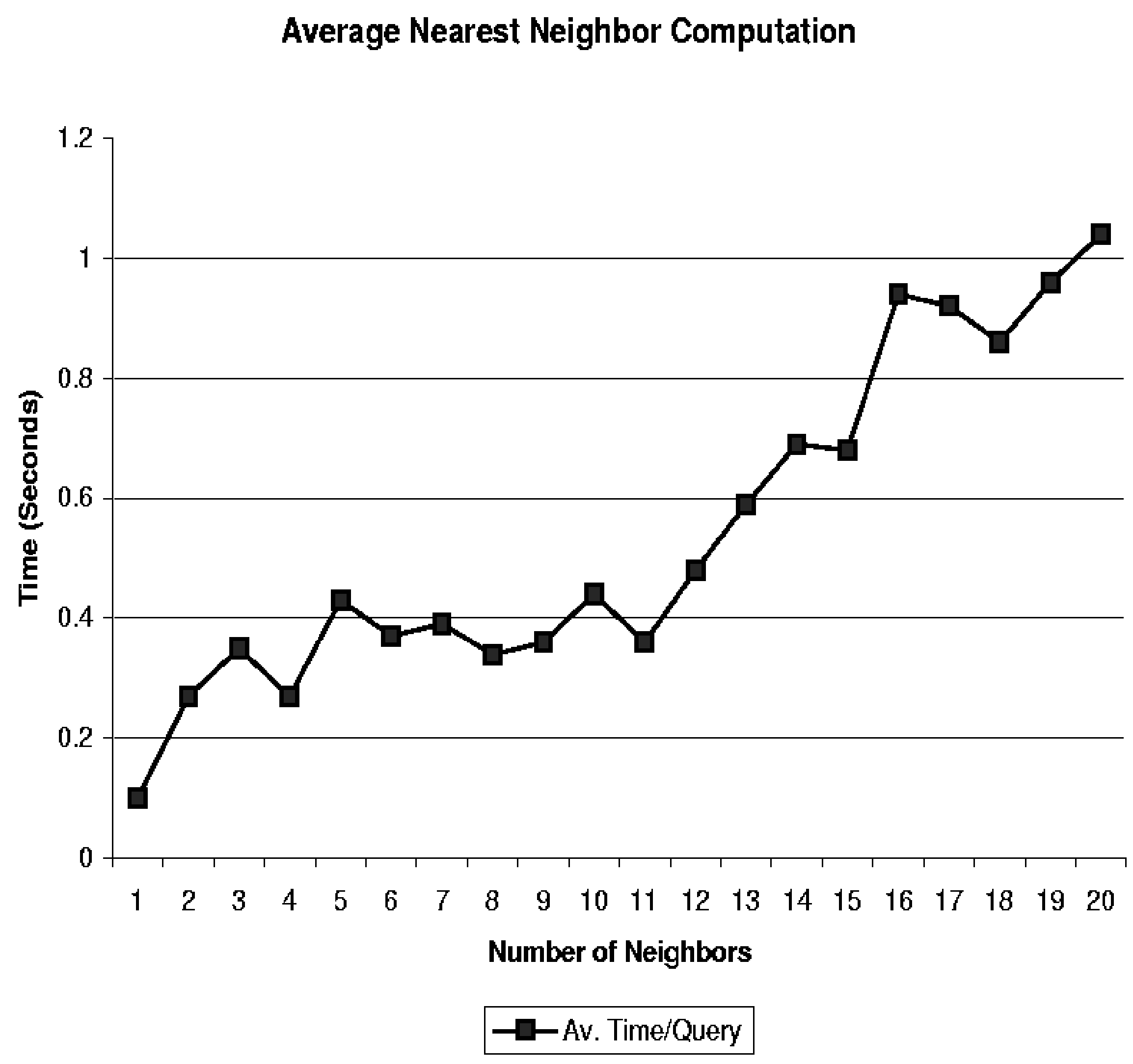


Figure 1.12: Performance of *k*-nearest neighbor algorithm, Average Time

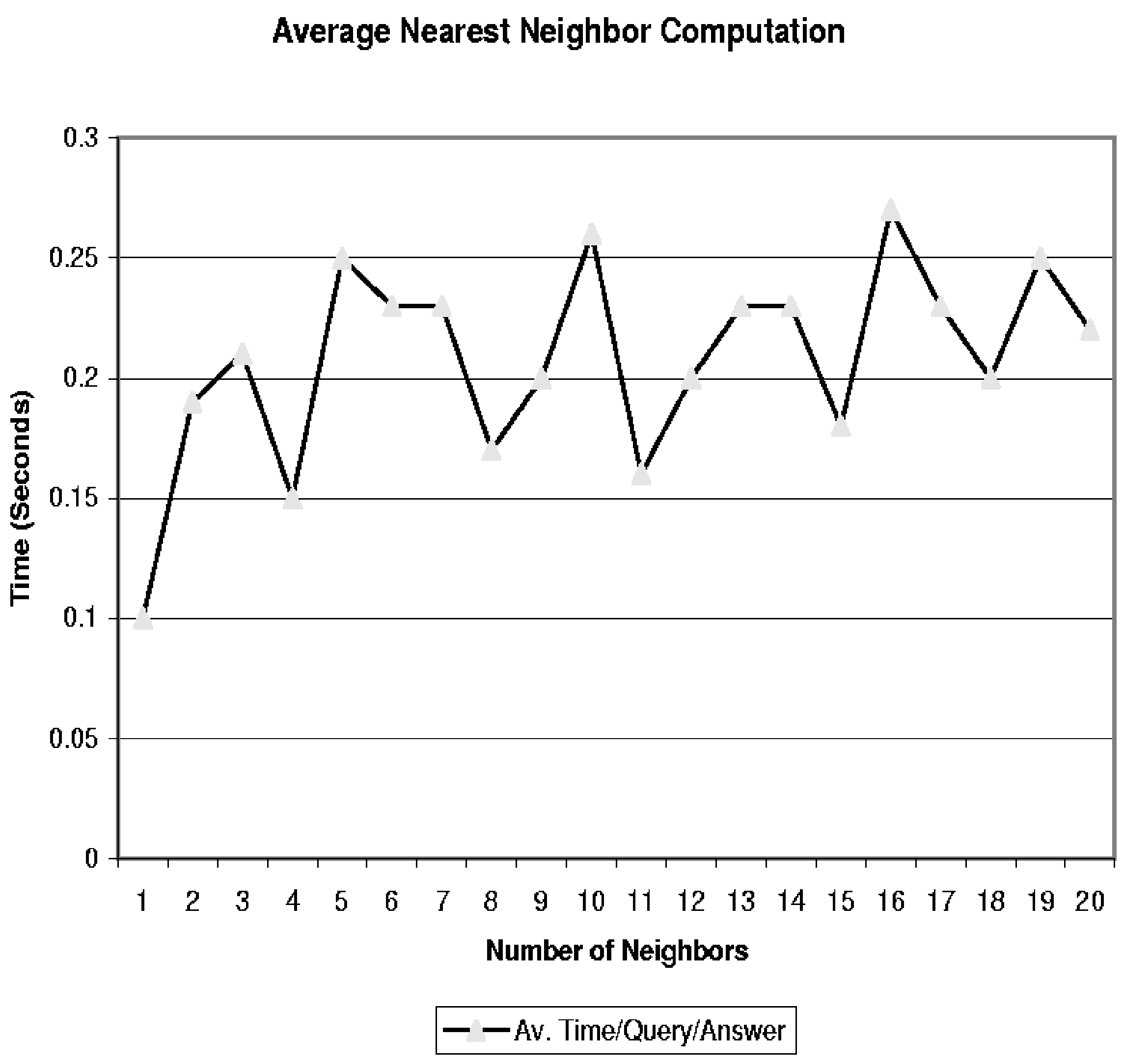


Figure 1.13: Performance of *k*-nearest neighbor, Average time per answer

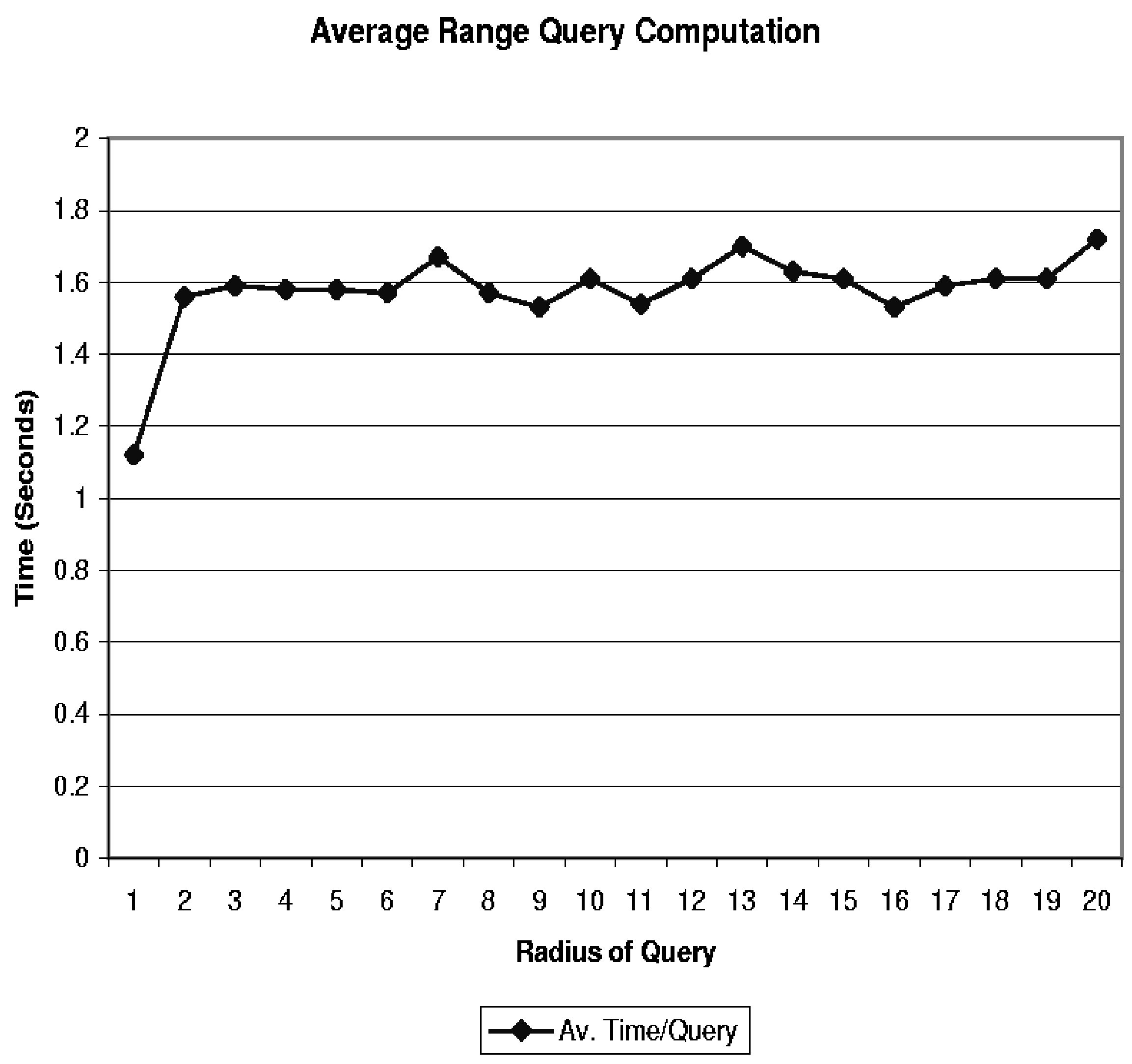


Figure 1.14: Performance of range query algorithm, Average Time

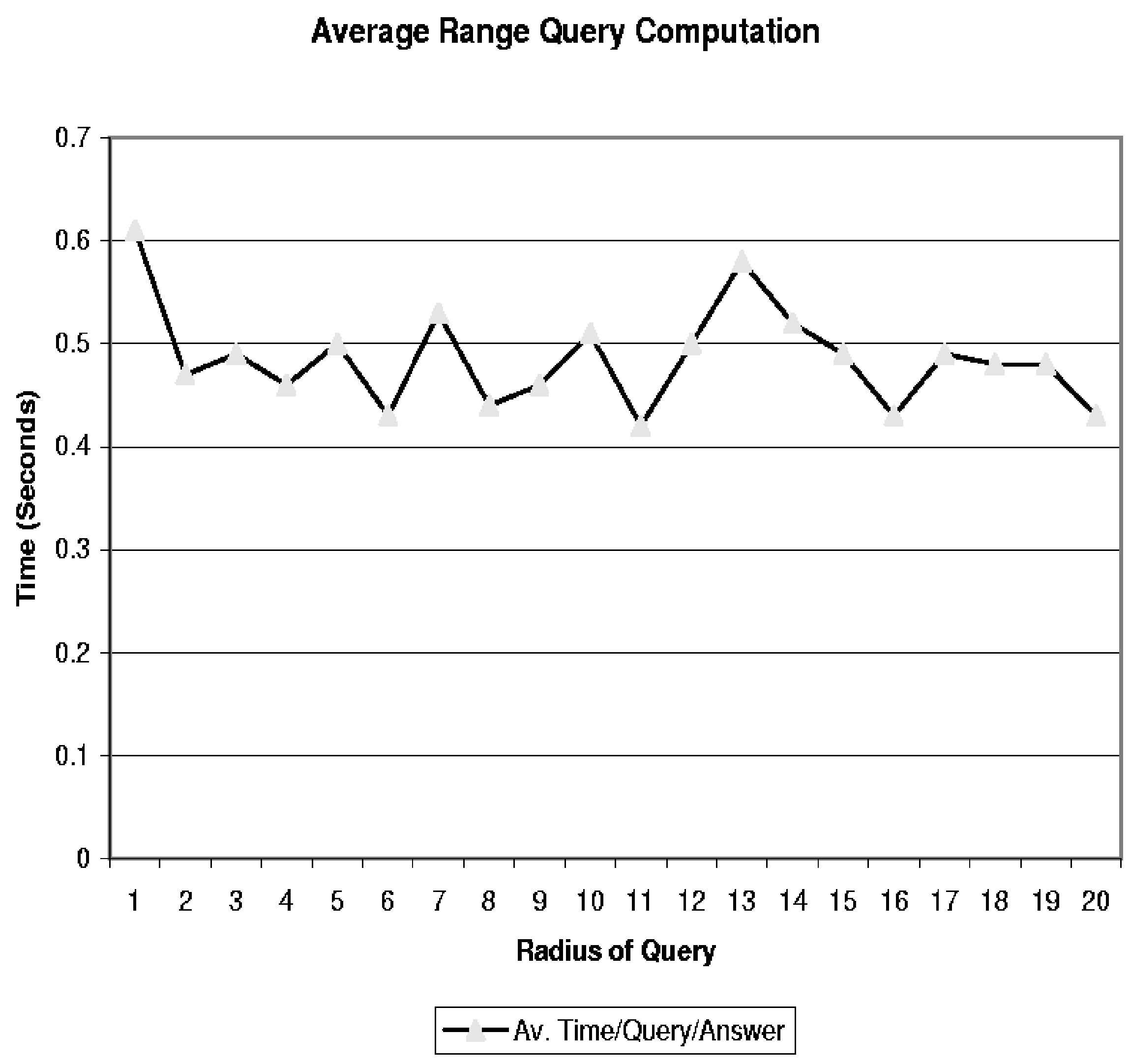
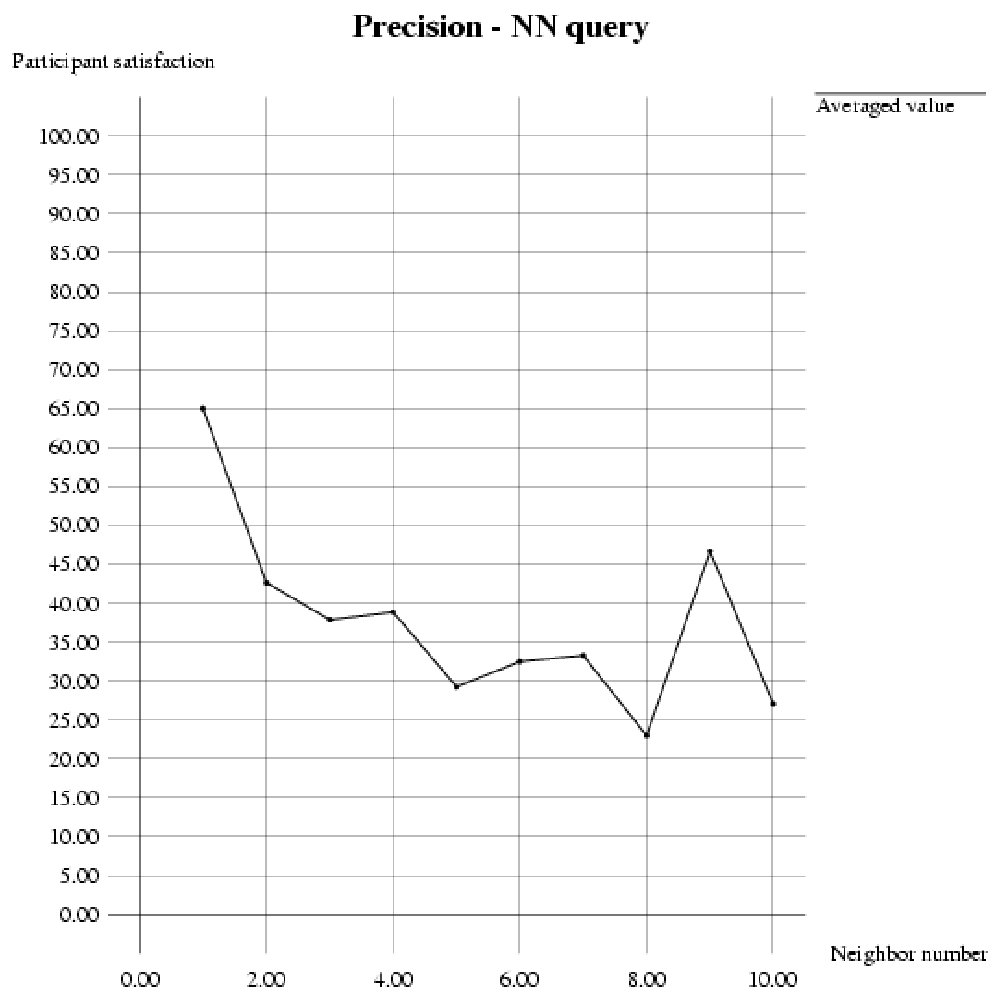


Figure 1.15: Performance of range query algorithm, Average time per answer

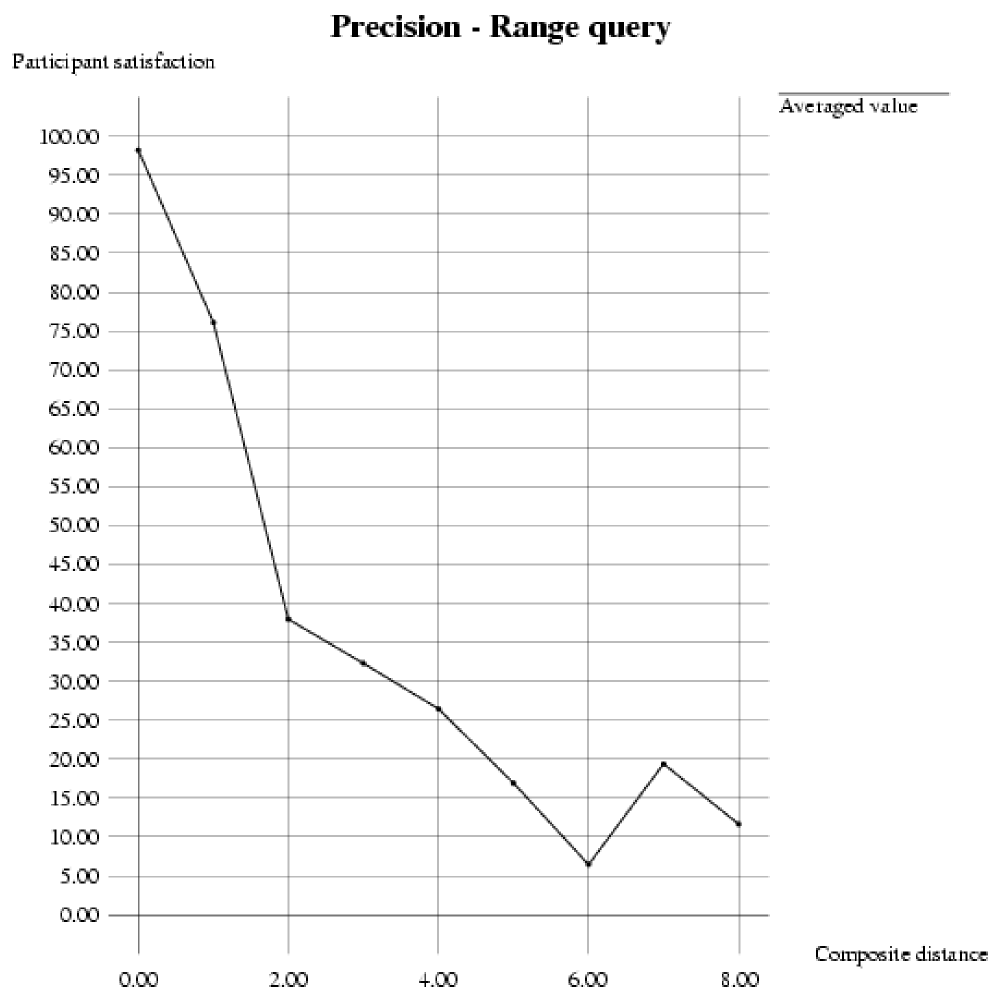
Quality of Returned Matches

We conducted an experiment involving 35 participants:

1. We used a simple verb hierarchy (10 nodes), noun-term hierarchy (90 nodes), and ServiceTable (100 services).
2. After an initial training phase, participants entered a precision phase where they were asked to perform 10 nearest neighbor and 10 range queries of their choice.
3. After each query result, participants typed in a ranking between 0 (least satisfied) and 100 (most satisfied).
4. Average satisfaction for nearest neighbor and range queries are shown below.



Precision for the *k*-nearest neighbor algorithm

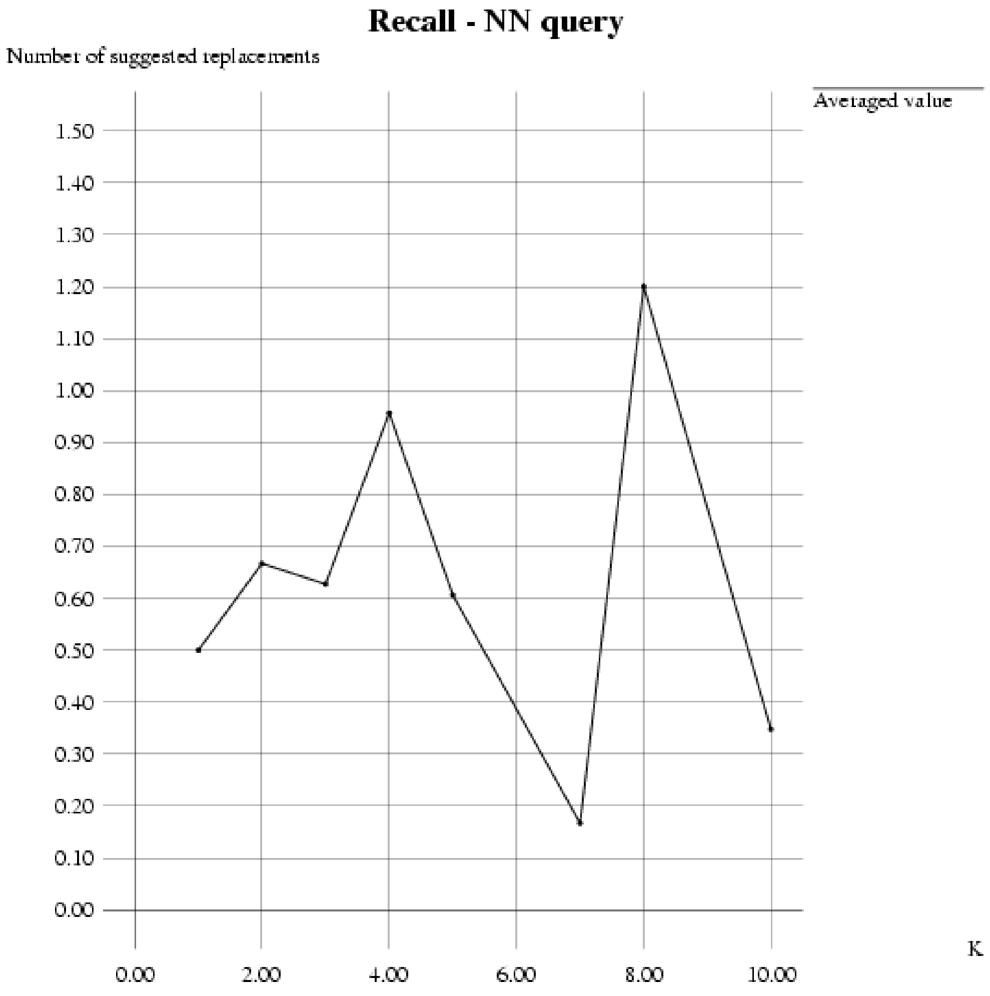


Precision for the range algorithm

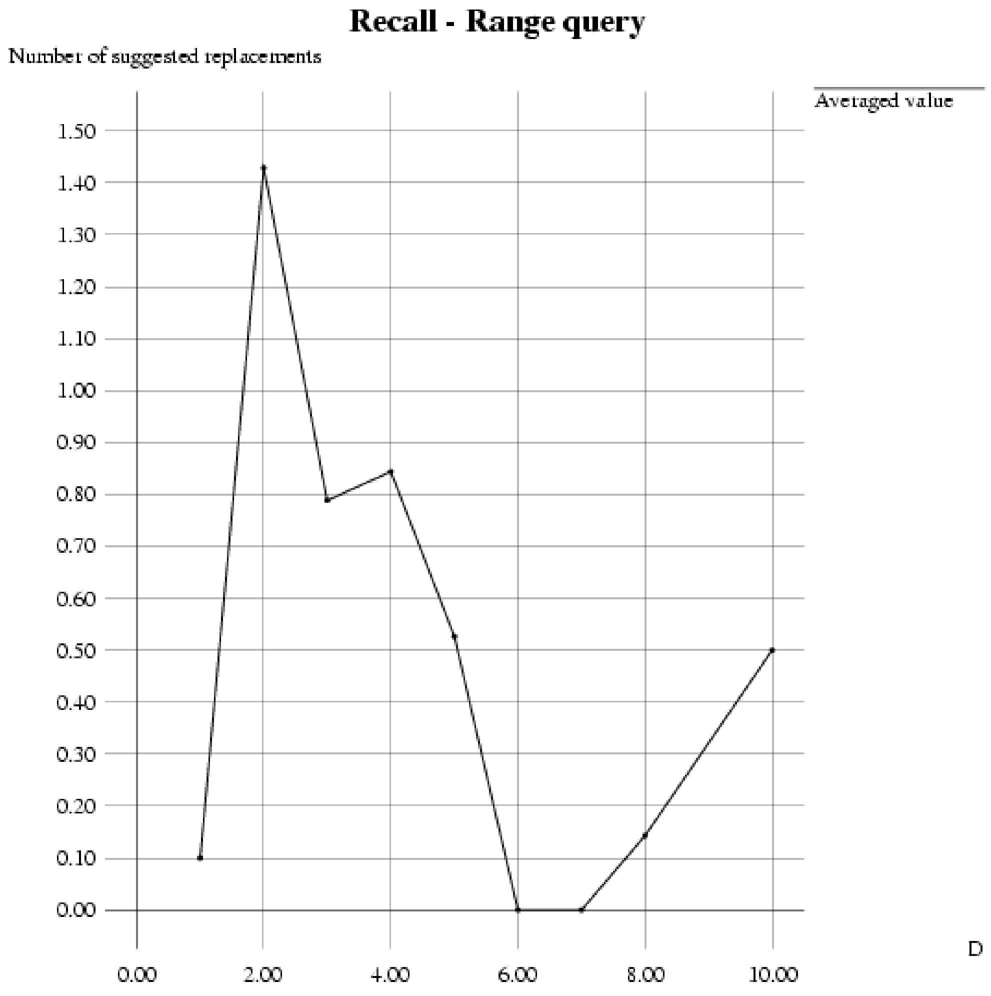
Figure 1.16: Experimental Results of Precision of our Algorithms

After completing the precision phase, participants started the **recall phase**.

1. They were allowed to view the ServiceTable (which up to this point was not available to them).
2. Meanwhile, they were presented with text boxes containing the query answers they gave in the previous phase.
3. After each answer, they were instructed to type in the name of all services in ServiceTable which did not appear as a query result but which should have been returned as an answer.
4. The **average number of these “suggested replacements”** for nearest neighbor and range query answers are shown below.



Recall for the *k*-nearest neighbor algorithm



Recall for the range algorithm (radius *D*)

Figure 1.17: Experimental Results of Recall of our Algorithms

1.5 Summary

1. We introduced **three szenarios** where multi-agency is important.
2. We presented the main *IMPACT*-architecture.
3. Agents need to use **services of other agents**.
   (a) We do not assume that agents precisly know about services of other agents.
   (b) We defined a language where such requests can be formulated
   ($\rightsquigarrow$ **service description language SDL**).
   (c) We presented algorithms to find the best matches for a request
   ($\rightsquigarrow$ **find_nn**, **range**) .

References

Apt, K., H. Blair, and A. Walker (1988). Towards a Theory of Declarative Knowledge. In J. Minker (Ed.), *Foundations of Deductive Databases and Logic Programming*, pp. 89–148. Washington DC: Morgan Kaufmann.

Arens, Y., C. Y. Chee, C.-N. Hsu, and C. Knoblock (1993). Retrieving and Integrating Data From Multiple Information Sources. *International Journal of Intelligent Cooperative Information Systems 2*(2), 127–158.

Arisha, K., F. Ozcan, R. Ross, V. S. Subrahmanian, T. Eiter, and S. Kraus (1999, March/April). IMPACT: A Platform for Collaborating Agents. *IEEE Intelligent Systems 14*, 64–72.

Bayardo, R., et al. (1997). Infosleuth: Agent-based Semantic Integration of Information in Open and Dynamic Environments. In J. Peckham (Ed.), *Proceedings of ACM SIGMOD Conference on Management of Data*, Tucson, Arizona, pp. 195–206.

Brink, A., S. Marcus, and V. Subrahmanian (1995). Heterogeneous Multimedia Reasoning. *IEEE Computer 28*(9), 33–39.

Chawathe, S., et al. (1994, October). The TSIMMIS Project: Integration of Heterogeneous Information Sources. In *Proceedings of the 10th Meeting of the Information Processing Society of Japan*, Tokyo, Japan. Also available via anonymous FTP from host db.stanford.edu, file /pub/chawathe/1994/tsimmis-overview.ps.

Dix, J., S. Kraus, and V. Subrahmanian (2001). Temporal agent reasoning. *Artificial Intelligence to appear*.

Dix, J., M. Nanni, and V. S. Subrahmanian (2000). Probabilistic agent reasoning. *Transactions of Computational Logic 1*(2).

Dix, J., V. S. Subrahmanian, and G. Pick (2000). Meta Agent Programs. *Journal of Logic Programming 46*(1-2), 1–60.

Eiter, T., V. Subrahmanian, and T. J. Rogers (2000). Heterogeneous Active Agents, III: Polynomially Implementable Agents. *Artificial Intelligence 117*(1), 107–167.

Eiter, T. and V. S. Subrahmanian (1999). Heterogeneous Active Agents, II: Algorithms and Complexity. *Artificial Intelligence 108*(1-2), 257–307.

Genesereth, M. R. and S. P. Ketchpel (1994). Software Agents. *Communications of the ACM 37*(7), 49–53.

Rogers Jr., H. (1967). *Theory of Recursive Functions and Effective Computability*. New York: McGraw-Hill.

Subrahmanian, V., P. Bonatti, J. Dix, T. Eiter, S. Kraus, F. Özcan, and R. Ross (2000). *Heterogenous Active Agents*. MIT-Press.

Wiederhold, G. (1993). Intelligent Integration of Information. In *Proceedings of ACM SIGMOD Conference on Management of Data*, Washington, DC, pp. 434–437.

Wilder, F. (1993). *A Guide to the TCP/IP Protocol Suite*. Artech House.