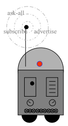
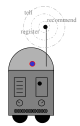
Theory and Practice of Agent Communication Languages

Tim Finin
University of Maryland
Baltimore County

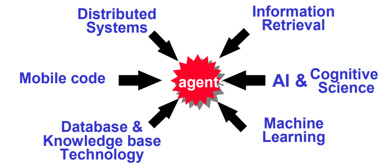
Yannis Labrou
Fujitsu Laboratories of America





Introduction to Agents & Agent Communication

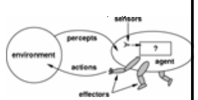
Agents: A system-building paradigm



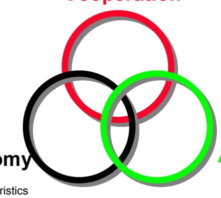
$$\frac{\text{agents}}{2001} = \frac{\text{objects}}{1982} = \frac{\text{structured programming}}{1974}$$

So, what's a software agent?

- No consensus yet, but several key properties are important to this emerging paradigm. Agents are:
  - **Autonomous,** taking the initiative as appropriate
  - **Goal-directed**, maintaining an agenda of goals which it pursues until accomplished or believed impossible.
  - **Taskable:** one agent can delegate rights/actions to another.
  - **Situated** in an environment (computational and/or physical) which it is aware of and reacts to.
  - **Cooperative** with other agents (software or human) to accomplish its tasks.
  - **Communicative** with other or software)
  - **Adaptive**, modifying beliefs behavior based on experience



Software Agent Characteristics



Note: these characteristics are not independent and, in general, support one another.

after Hyacinth Nwana, 1996

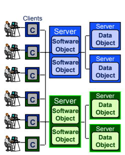
Agent Architectures

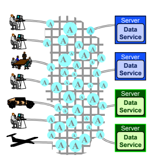
People are using several architectures for agent-based information systems.

- Mediated architectures
- Multi-agent systems
- Markets and swarms

Mediated Architectures

- Agents generalize the client-server architecture which has dominated the Internet since its beginning
- Wiederhold introduced the notion of a "mediated architecture" for information systems





Multi-agent Systems

- Some research focuses on developing sophisticated individual agents with advanced capabilities.
- Other research is focused on multi-agent systems (MAS) with an emphasis on
  - agent-to-agent communication
  - cooperation and collaboration
  - team and coalition formation
  - information sharing among the team
  - joint beliefs, goals and plans

Agent markets and swarms

- Yet another architectural view is the decentralized market or swarm.
- Key idea -- the parallel, autonomous actions of a large collection of individual agents results in *emergent behavior* of the collective.
- The *market view* usually assumes rational agents whereas the *swarm view*, associated with artificial life, does not.

Some key ideas

- Software agents offer a new paradigm for very large scale *distributed heterogeneous applications*.
- The paradigm focuses on the *interactions* of autonomous, cooperating processes which can adapt to humans and other agents.
- Mobility is an orthogonal characteristic which many, but not all, consider central.
- Intelligence is always a desirable characteristic but is not required by the paradigm.
- The paradigm is still forming.

Why is communication important?

- Most, but not all, would agree that communication is a requirement for cooperation.
- Societies can do things that no individual (agent) can.
- Diversity introduces heterogeneity.
- Autonomy encourages disregard for other agents' internal structure.
- Communicating agents need only care about understanding a "common language".

What is communication?

- Communication almost always means "communication in a common language"
- "Language" does not include natural languages only.
- Understanding a "common language" means:
  - understanding of its vocabulary, i.e., understanding of the meaning of its tokens
  - knowing how to effectively use the vocabulary to perform tasks, achieve goals, effect one's environment, etc.
- For software agents, an Agent Communication Language (ACL) is primarily concerned with the vocabulary

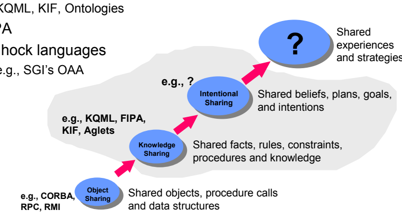
Agent Communication Languages: Useful Concepts

Agent Communication

- Agent-to-agent communication is key to realizing the potential of the agent paradigm, just as the development of human language was key to the development of human intelligence and societies.
- Agents use an ***Agent Communication Language*** or ACL to communication information and knowledge.
- Genesereth (CACM, 1992) defined a software agent as any system which uses an ACL to exchange information.

Some ACLs

- Is CORBA an ACL?
- Knowledge sharing approach
  - KQML, KIF, Ontologies
- FIPA
- Ad hock languages
  - e.g., SGI's OAA



To communicate is to manipulate a "common language"

- Effective agent communication involves two aspects:
  - possessing the understanding of a "common language", as humans do for various domains and tasks
  - using the common language in order to achieve tasks and goals, and to effect an agent's environment
- The understanding of the meaning of the tokens of a language is the substrate for any form of communication.
- Understanding the tokens alone, does not mean ability to communicate; the use of (any) language is driven by a purpose.

Agent Communication, at the technical level

- Messages are transported using some lower-level transport protocol (SMTP,TCP/IP, HTTP, IIOP, etc.)
- An Agent Communication Language (ACL) defines the types of messages (and their meaning) that agents may exchange.
- Over time, agents engage in "conversations." Such interaction protocols (negotiation, auction, etc.), defines task-oriented, shared sequences of messages.
- Some higher-level conceptualization of an agent's goals and strategies drives the agent's communicative (and non-communicative) behavior.

What Comes Next

- Conceptual and theoretical foundations I
  - The layered nature of communication, services, mobility issues, Speech Act Theory, BDI
- Conceptual and theoretical foundations II
  - Knowledge Representation and Ontology Issues
- The Knowledge Sharing Effort
  - KIF, KQML, Ontolingua
- The Foundation for Intelligent Physical Agents
  - FIPA ACL, FIPA Agent Platform
- Semantic accounts for ACLs
- Alternative approaches and languages
- APIs, Systems and Applications
- Trends and future directions
- Conclusions

Conceptual and Theoretical Foundations I

Historical Note: Knowledge Sharing Effort

- Initiated by DARPA circa 1990
- Sponsored by DARPA, NSF, AFOSR, etc.
- Participation by dozens of researchers in academia and industry.
- Developing techniques, methodologies and software tools for *knowledge sharing* and *knowledge reuse*.
- Sharing and reuse can occur at *design*, *implementation* or *execution* time.

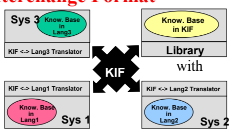
Knowledge Sharing Effort

- Knowledge sharing requires a communication which requires a common language
- We can divide a language into syntax, semantics, and pragmatics
- Some existing components that can be used independently or together:
  - **KIF** - knowledge Interchange Format *(syntax)*
  - **Ontolingua** - a language for defining sharable ontologies *(semantics)*
  - **KQML** - a high-level interaction language *(pragmatics)*

Propositional

Propositional attitudes

Knowledge Interchange Format



- KIF ~ First order logic set theory
- An **interlingua** for encoded declarative knowledge
  - Takes translation among n systems from $O(n^2)$ to $O(n)$
- Common language for reusable knowledge
  - Implementation independent semantics
  - Highly expressive - can represent knowledge in typical application KBs.
  - Translatable - into and out of typical application languages
  - Human readable - good for publishing reference models and ontologies.
- Current specification at http://logic.stanford.edu/

Common Semantics Shared Ontologies and Ontolingua

- ***Ontology***: A common vocabulary and agreed upon meanings to describe a subject domain.
- Ontolingua is a language for building, publishing, and sharing ontologies.
  - A web-based interface to a browser/editor server.
  - Ontologies can be automatically translated into other content languages, including KIF, LOOM, Prolog, etc.
  - The language includes primitives for combining ontologies.

Common Pragmatics

Knowledge Query and Manipulation Language

- KQML is a high-level, message-oriented, communication language and protocol for information exchange independent of content syntax and ontology.
- KQML is also independent of
  - transport mechanism, e.g., tcp/ip, email, corba, IIOP, ...
  - High level protocols, e.g., Contract Net, Auctions, …
- Each KQML message represents a single *speech act* (e.g., ask, tell, achieve, …) with an associated semantics and protocol.
- KQML includes primitive message types of particular interest to building interesting agent architectures (e.g., for mediators, sharing intentions, etc.)

Common High-level Protocols

- There is also a need for communication agents to agree on the agent-level protocols they will use.
- The protocol is often conveyed via an extra parameter on a message
  - (ask :from Alice :to Bob … **:protocol auction42** …)
- Common protocols:
  - Contract net
  - Various auction protocols
  - Name registration
- These protocols are often defined in terms of constraints on possible conversations and can be expressed as
  - Grammars (e.g., DFAs, ATNs, …)
  - Petri networks
  - Conversation plans
  - Rules or axioms

Common Service Infrastructure

- Many agent systems assume a common set of services such as:
  - Agent Name Sever
  - Broker or Facilitator
  - Communication visualizer
  - Certificate server
- These are often tied rather closely to an ACL since a given service is implemented to speak a single ACL
- Moreover, some of the services (e.g., name registration) may be logically ACL-dependent
  - e.g., Some ACLs don't have a notion of an agent's name and others have elaborate systems of naming

Speech Act Theory and BDI Theories

The intentional level, BDI theories, speech acts and ACLs: How do they all fit together?

- ACL have message types that are usually modeled after speech acts
- Speech acts may be understood in terms of an intentional-level description of an agent
- An intentional description makes references to beliefs, desires, intentions and other modalities
- BDI frameworks have the power to describe an agents' behavior, including communicative behavior

The intentional stance

- Agents have "propositional attitudes"
- Propositional attitudes are three-part relationship between
  - an agent,
  - a content-bearing proposition (e.g., "*it is raining*"), and
  - a finite set of propositional attitudes an agent might have with respect to the proposition (e.g., believing, asserting, fearing, wondering, hoping, etc.)
- <*a*, fear, raining(t_{now}) >

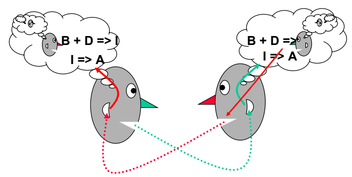
On ascribing mental qualities to machines

- The issue is not whether a system is really intentional but whether we can coherently view it as such (Daniel Dennett)
- Ascribing mental qualities to machines (John McCarthy):
  - **legitimacy**: the ascription expresses the same information about a machine that it expresses about a person
  - **usefulness**: the ascription helps us understand the structure of the machine, its past or future behavior, or how to repair it or improve it.

BDI Agents, Theories and Architectures

- BDI architectures describe the internal state of an agent by the mental states of beliefs, goals and intentions
- BDI theories provide a conceptual model of the knowledge, goals, and commitments of an agent
- BDI agents have some (implicit or explicit) representations of the corresponding attitudes

BDI Model and Communication



- Communication is a means to (1) reveal to others what our BDI state is and (2) attempt to effect the BDI state of others.
- Note the recursion: an agent has beliefs about the world, beliefs about other agents, beliefs about the beliefs of other agents, beliefs about the beliefs another agent has about it, ...

Criticism of BDI theories

- The necessity of having all three modalities is questioned from both ends:
  - too few
  - too many
- System builders question their relevance in practice:
  - multi-modal BDI logics do not have complete axiomatizations
  - they are not efficiently computable
- There is a gap between theory and practice

Speech Act Theory

High level framework to account for human communication

Language as Action (Austin)

- Speakers do not just utter true or false sentences
- Speakers perform speech acts:

  requests, suggestions, promises, threats, etc.
- Every utterance is a speech act

Speech Act Theory (continued)

Example: *"Shut the door !"*

- **locution**

  physical utterance with context and reference, *i.e.*, who is the speaker and the hearer, which door etc.
- **illocution**

  the act of conveying intentions, *i.e.*, speaker wants the hearer to close the door
- **perlocutions**

  actions that occur as a result of the illocution, *i.e.*, hearer closes the door

Conceptual and Theoretical Foundations II

Representation and Reasoning

- Intelligent agents need to be able to represent and reason about many things, including:
  - models of other agents (human or artificial) beliefs, desires, intentions, perceptions, plans, etc.
  - task, task structures, plans, etc.
  - meta-data about documents and collections of documents
- In general, they will need to **communicate** the same range of knowledge.
- A variety of content languages have been used with ACLs, including KIF, SL, Loom, Prolog, CLIPS, SQL, …
- There is a special interest in content languages that can serve as a neutral, but expressive, **interlingua** for a wide range of systems.
- We'll look at KIF in a bit more detail.

KR Language Components

- **A logical formalism**
  - Syntax for well formed formulae (wffs)
  - Vocabulary of logical symbols (e.g., and, or, not, =>, ...)
  - Interpretation semantics for the logical symbols, e.g., "(=> A B)" is true if and only if B is true or A is false.
- **An ontology**
  - Vocabulary of non-logical symbols (relations,functions,constants)
  - Definitions of non-primitive symbols, e.g. (<=> (Bachelor ?x) (AND (Man ?x) (Unmarried ?x)))
  - Axioms restricting the interpretations of primitive symbols, e.g. (=> (Person ?x) (Gender (Mother ?x) Female))
- **A proof theory**
  - Specification of the reasoning steps that are logically sound, e.g. From "(=> S1 S2)" and "S1", conclude "S2"

Classical Definitions Are Not Enough

- Definitions provide equivalent expressions
  - $R(x) \equiv \Phi(x)$
  - E.g., bachelor(x) ≡ man(x) Λ ~married(x)
- Defined symbols can be eliminated by replacement
- KB is then expressed in terms of undefined symbols
- Undefined symbols are given "meaning" by axioms
  - E.g., ~[on(x,y) Λ on(y,x)]
- Thus, ontologies must have both definitions and axioms

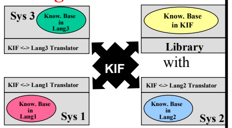
O-O Languages Too Restrictive

- Frames, object schema, description logics are popular KR languages used for ontologies
- They support definitional axioms of the form:
  - $R(x) \Rightarrow \ldots \wedge P(x) \wedge \ldots$ {subclass}
  - $R(x) \Rightarrow \ldots \wedge [S(x,y) \Rightarrow P(y)] \wedge \ldots$ {value class}
  - $R(x) \Rightarrow \ldots \wedge \exists y\ S(x,y) \wedge \ldots$ {value cardinality}

  …
- They don't support –
  - N-ary relations and functions
  - Standard properties of relations and functions
    E.g., transitive, symmetric
  - Partial sufficient conditions
    E.g., $x>0 \Rightarrow R(x)$

  …

Knowledge Interchange Format



- KIF ~ First order logic set theory
- An **interlingua** for encoded declarative knowledge
  - Takes translation among n systems from $O(n^2)$ to $O(n)$
- Common language for reusable knowledge
  - Implementation independent semantics
  - Highly expressive - can represent knowledge in typical application KBs.
  - Translatable - into and out of typical application languages
  - Human readable - good for publishing reference models and ontologies.
- Current specification at http://logic.stanford.edu/

Other alternatives

- OKBC (see ontologies)
- Java objects (see AgentBuilder)
- SL (see FIPA)
- Constraints
- Database tuples
- RDF
- DAML
- *..your favorite representation language here..*

Ontologies

Common Semantics
Shared Ontologies and Ontolingua

Ontology : A common vocabulary and agreed upon meanings to describe a subject domain.

On*tol"o*gy (?), n. [Gr. the things which exist (pl.neut. of , , being, p.pr. of to be) + -logy: cf.F. ontologie.]

That department of the science of metaphysics which investigates and explains the nature and essential properties and relations of all beings, as such, or the principles and causes of being.

Webster's Revised Unabridged Dictionary (G & C. Merriam Co., 1913, edited by Noah Porter)

This is not a profoundly new idea …
- Vocabulary specification
- Domain theory
- Conceptual schema (for a data base)
- Class-subclass taxonomy
- Object schema

Importance of ontologies in communication

- An example of the importance of ontologies in communication is the fate of NASA's Mars Climate Orbiter
- It crashed into Mars on September 23, 1999
- JPL used metric units in their program controlling the thrusters and Lockheed-Martin used imperial units.
- Instead of establishing an orbit at an altitude of 140km, it did so at 60km, causing it to burn up in the Martian atmosphere.

Conceptual Schemas

A conceptual schema specifies the intended meaning of concepts used in a data base

Data Base:

139	
74.50	
140	

77.60

Data Base Schema:

Table: price
*stockNo: integer; cost: float

Conceptual Schema:

price(x, y) =>
∃ (x', y') [auto_part(x')
& part_no(x') = x
& retail_price(x', y', Value-Inc)
& magnitude(y', US_dollars) = y]

Auto Product Ontology

Product Ontology

Units & Measures Ontology

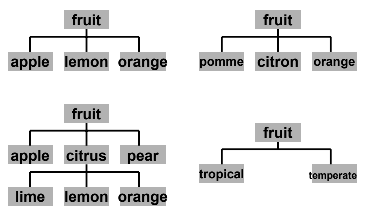
Implicit vs. Explicit Ontologies

- Systems which communicate and work together must share an ontology.
- The shared ontology can be **implicit** or **explicit**.
- Implicit ontology are typically represented only by procedures
- Explicit ontologies are (ideally) given a declarative representation in a well defined knowledge representation language.

Conceptualizations, Vocabularies and Axiomitization

- Three important aspects to explicit ontologies
  - **Conceptualization** involves the underlying model of the domain in terms of objects, attributes and relations.
  - **Vocabulary** involves assigning symbols or terms to refer to those objects, attributes and relations.
  - **Axiomitization** involves encoding rules and constraints which capture significant aspects of the domain model.
- Two ontologies may
  - be based on different conceptualizations
  - be based on the same conceptualization but use different vocabularies
  - differ in how much they attempt to axiomitize the ontologies

Simple examples



Ontologies vs. KBs

Ontologies are distinguished from KBs not by their form, but by the role they play in representing knowledge

- Consensus models for a domain
- Emphasis on properties that hold in all situations
- Emphasis on classes rather than instances
- Intended to support multiple tasks and methods
- Don't change during problem solving and are suited for "compiling" into tools
- Need to satisfy a community of use
  - Emphasis on collaborative development
  - Emphasis on translation to multiple logical formalisms
- Useful for education

Big Ontologies

- There are several large, general ontologies that are freely available.
- Some examples are:
  - **Cyc** - Original general purpose ontology
  - **WordNet** - a large, on-line lexical reference system
  - **World Fact Book --** 5Meg of KIF sentences!
  - **UMLS** - NLM's Unified Medical Language System
- See http://www.cs.utexas.edu/users/mfkb/related.html for more

Ontology Conclusions

- Shared ontologies are essential for agent communication and knowledge sharing
- Ontology tools and standards are important
  - Ontolingua and OKBC are good examples
  - XML, RDF, DAML may be a next step
- Some large general ontologies are available
  - Cyc, WFB, WordNet, …
- For more information…
  - http://www.kr.org/top describes projects addressing major ontology construction issues
  - Ontology mailing list: send mail to majordomo@cs.umbc.edu with "*info ontology*" in message body for information.
  - ANSI Ad Hoc Group on Ontology Standards: http://WWW-KSL.Stanford.EDU/onto-std/

Knowledge Sharing Effort (KSE)

Knowledge Interchange Format (KIF)

KIF Syntax and Semantics

- Extended version of first order predicate logic
- Simple list-based linear ASCII syntax, e.g.,
  - (forall ?x (=> (P ?x) (Q ?x)))
  - (exisits ?person (mother mary ?person))
  - (=> (apple ?x) (red ?x))
  - (<<= (father ?x ?y) (and (child ?x ?y) (male ?x))
- Model-theoretic semantics
- KIF includes an axiomatic specification of large function and relation vocabulary and a vocabulary for numbers, sets, and lists

Implications and Rules

- KIF distinguishes between implications and rules.
- Implication (e.g., (=> (p ?x) (q ?x)) is a connective
- Rules are directed (forward or backward)
  - (=>> (p ?x) (q ?x)) vs. (<< = (q ?x) (p ?x))
- Rules involve derivation. E.g., from
  - (<<= (status-known ?x) (citizen ?x))
  - (<<= (status-known ?x) (not (citizen ?x)))
- we infer (status-known Joe) only if one of (citizen Joe) or (not (citizen Joe)) can be inferred.

Functions and Relations

- Functions and relations are sets of lists in the universe of discourse
  - (= r (setofall (listof n1 … nk) (r n1 … nk)))
- They can be arguments to other functions & relations
  - E.g., (transitive R), (inverse R1 R2), (one-one F), (range F)
- The can be "applied" to arguments
  - (holds ?r 1 2)
  - (value ?f 1 2)
  - (<=> (transitive ?r)
    - (=> (and (holds ?r ?x ?y) (holds ?r ?y ?z)) (holds ?r ?x ?z))))

Big KIF and Little KIF

- That KIF is highly expressive language is a desirable feature; but there are disadvantages.
  - complicates job of building fully conforming systems.
  - resulting systems tend to be "heavyweight"
- KIF has "*conformance categories"* representing dimensions of conformance and specifying alternatives within that dimension.
- A *"conformance profile"* is a selection of alternatives from each conformance category.
- System builders decide upon and adhere to a conformance profile sensible for their applications.

KIF vs ANSI KIF

- KIF is the object of an ANSI Ad Hoc standardization group (X3T2)
- ANSI KIF is somewhat different from previous specs
  - No non-monotonic rules
  - Allow for possible (future) higher-order extensions
  - Defines a standard infix format for presenting KIF

KIF Software

- Several KIF-based reasoners in LISP are available from Stanford (e.g., EPILOG).
- IBM's ABE (Agent Building Environment) & RAISE reasoning engine use KIF as their external language.
- Stanford's Ontolingua uses KIF as its internal language.
- Translators (partial) exist for a number of other KR languages, including LOOM, Classic, CLIPS, Prolog,...
- Parsers for KIF exist which take KIF strings into C++ or Java objects.

KIF Summary

- KIF is the only widely used interlingua for KB systems
  - KIF is the focus of an ANSI standardization effort
  - See KIF spec at <http://logic.stanford.edu/> and also <http://www.cs.umbc.edu/kif> for more information.
- Its future outside the AI-related community is unclear
  - It may not be acceptable to a wider community because its too logic-oriented or not object-oriented or …
  - Then again, it's expressive power may win the day!
- Defining a mapping of KIF to XML might make it more acceptable.

Knowledge Query and Manipulation Language (KQML)

KQML

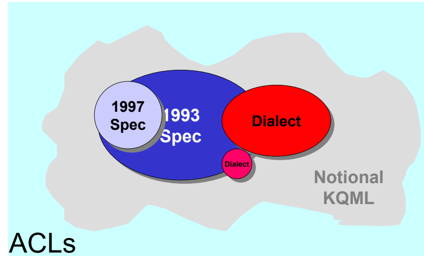
Knowledge Query and Manipulation Language

- KQML is a high-level, message-oriented, communication language and protocol for information exchange independent of content syntax and ontology.
- KQML is independent of
  - the transport mechanism (e.g., tcp/ip, email, corba objects, IIOP, etc.)
  - Independent of content language (e.g., KIF, SQL, STEP, Prolog, etc.)
  - Independent of the ontology assumed by the content.
- KQML includes primitive message types of particular interest to building interesting agent architectures (e.g., for mediators, sharing intentions, etc.)

KQML Specifications

- There are two KQML specification documents:
  - *Specification of the KQML Agent-Communication Language plus example agent policies and architectures*, The DARPA Knowledge Sharing Initiative, External Interfaces Working Group, 1993. http://www.cs.umbc.edu/papers/kqml93.pdf
  - *A Proposal for a new KQML Specification*, Yannis Labrou and Tim Finin, TR CS-97-03, Feb.1997, Computer Science and Electrical Engineering Department, University of Maryland Baltimore County, Baltimore, MD 21250. http://www.cs.umbc.edu/kqml/papers/kqml97.pdf
- There are also many dialects and “extended” versions of KQML plus lots of important concepts not addressed in either specification document (e.g., security).
- We’ll mostly focus on the 1997 document plus other ideas used in practice.

Multiple KQML dialects



A KQML Message

performative
parameter
value

(***tell*** :sender bhkAgent
:receiver fininBot
:in-reply-to id7.24.97.45391
:**ontology** ecbk12
:language Prolog
:content
“price(ISBN3429459,24.95)”)

Represents a single *speech act* or *performative*

ask, tell, reply, subscribe, achieve, monitor, ...

with an associated *semantics and protocol*

tell(i,j, $B_i\phi$) = fp[$B_i B_i\phi \wedge \neg B_i(Bif_j B_i\phi \vee Uif_j B_i\phi)$] $\wedge$ re[$B_j B_i\phi$] ...

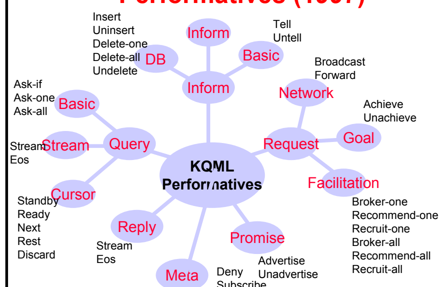
and a list of *attribute/value pairs*

:content, :language, :from, :in-reply-to

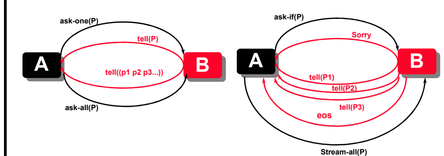
KQML Syntax

- KQML was originally defined as a language with a particular linear syntax which is based on Lisp.
- Alternate syntaxes have been used, e.g., based on SMTP, MIME, HTTP, etc.)
  - There are proposals for a meta-syntax that can support different syntactic dialects.
- KQML has also been mapped onto objects and passed from agent to agent as objects (e.g., if in the same memory space) or serialized objects.
- **KQML is not about syntax.**

Performatives (1997)

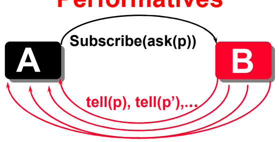


Simple Query Performatives



- The **ask-one**, **ask-all**, **ask-if**, and **stream-all** performatives provide a basic query mechanism.

Active Information Performatives



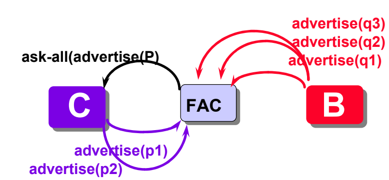
- The ***subscribe*** performatives is used to request active information services.
- **subscribe(P)** means roughly *“Keep your response to P current”*
- Note that it’s content is an embedded KQML performative and thus it’s *:language* is *KQML*

Facilitation Services

Facilitators are a class of agents who

- traffic in meta-knowledge about other agents.
- provide communication services such as:
  - message forwarding and broadcasting
  - resource discovery
  - matchmaking
  - content-based routing
  - meta-knowledge queries
- Performatives of special interest to facilitators are
  - advertise, broker, recruit, recommend, forward, broadcast, etc.
- Brokers are generally considered to focus on matchmaking
- Facilitators can be intelligent or not
  - Intelligent facilitators use domain knowledge in matching services needs and offers.

Capability Description

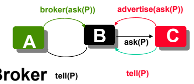


The **advertise** performative is used to describe the performatives an agent is prepared to accept.

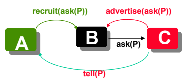
Facilitation Performatives

The three facilitation performatives come in a X-one and X-all versions:

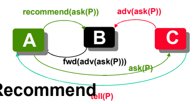
- Broker-one and broker-all
- Recruit-one and recruit-all
- recommend-one and recommend-all



Broker



Recruit



Recommend

Agent Names

- System for mapping agents into names is important in most ACLs
- KQML assumes that names are *local*
  - A can register with B under the name Alice
  - A can register with C under the name Albert
- Doesn't preclude the use of a central *Agent Name Server,* an architecture used by most systems
- What gets registered under a name? Contact information like:
  - name(albert, tcpip, [cujo.cs.umbc.edu,8080]).
  - Name(albert,smtp,[agenta@agents.umbc.edu)
  - name(albert,http,[www.agents.umbc.edu:8090/kqml/albert])

KQML Semantics

- **Myth:** KQML doesn't have a good semantic description.
- **Reality:** This was true for the first few years of its use, but has not been true since 1994.
- **Yannis Labrou** defined a semantics in
  - Yannis Labrou and Tim Finin, *A semantics approach for KQML -- a general purpose communication language for software agents*, Third International Conference on Information and Knowledge Management (CIKM'94), Nov. 1994.
  - Yannis Labrou, *Semantics for an Agent Communication Language*, Ph.D. Thesis, UMBC, 1996.
  - Yannis Labrou and Tim Finin, *Semantics and Conversations for an Agent Communication Language*, in "Readings in Agents", Michael Huhns and Munindar Singh (editors.), Morgan Kaufmann, 1997. (reprint of IJCAI-97 paper).
- Other approaches to defining the semantics have been partially explored (more on this later).

KQML APIs and System Interfaces

- There have been dozens of APIs written for KQML
- Written in and for different languages
  - Lisp, Scheme, Prolog, C/C++, Java, CLIPS, Smalltalk, Tcl, Perl, ...
- And interfacing to may different systems
  - Loom, Cyc, SIMS (Information Integration), SIPE (Planning), Various Databases, …
- More recent is the appearance of KQML-speaking "*agent shells*", offering more than just an API.
- More on these later

For More Information

- Mailing lists
  - kqml@cs.umbc.edu
  - send email to *majordomo@cs.umbc.edu* with body text "info kqml" for more information
- Information server
  - http://www.cs.umbc.edu/kqml/
- Specification documents
  - 1993: http://www.cs.umbc.edu/kqml/kqml93.pdf
  - 1996: http://www.cs.umbc.edu/kqml/kqml97.pdf

Ontolingua

Ontolingua - Language

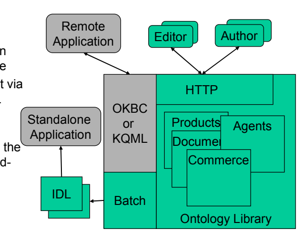
- Ontolingua allows full KIF
  - 1st order logic with relation constants in domain of discourse
  - Extremely expressive
  - Too much for most users
  - Too much for most systems!
- Ontolingua provides an object-oriented projection
- Statements within the o-o sublanguage easy to make
  - But any statement is allowed
- Ontolingua separates representation from presentation

Ontolingua - Library

- Library of modules supports reuse
- Authors assemble a new ontology
  - Assembly defines a general graph
  - Cycles are allowed (sports and medicine)
- Authors may augment definitions
  - But you can never say less!
  - Different authors may make incompatible extensions

Ontolingua - Architecture

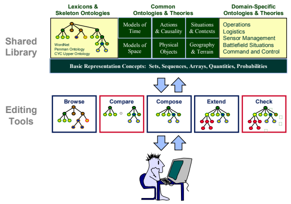
- Authors, editors, reviewers interaction via the web interface
- Applications interact via the OKBC or KQML interface
- Batch translation of ontologies supports the construction of stand-alone applications



Ontology Library and Editing Tools

Ontolingua is a language for building, publishing, and sharing ontologies.

- A web-based interface to a browser/editor server at *http://ontolingua.stanford.edu/* and mirror sites.
- Ontologies can be translated into a number of content languages, including KIF, LOOM, Prolog, CLIPS, etc.



Ontolingua - Usage

- Ontolingua is (one of) the most widely used knowledge development environments
  - Available since 1/94 at http://ontolingua.stanford.edu
  - Over 4500 total users, 1200 current users, 300 active users
  - Over 4,200,000 user commands executed
  - Recently averaging over 7000 commands per day
  - Over 800 ontologies stored on the KSL server
  - Mirror sites in Spain, Netherlands, UMBC, and corporate sites
- Applications include
  - Enterprise modeling, electronic commerce, engineering, ribosomal structure modeling, workflow modeling, molecular biology, cross-disciplinary design and simulation, drug interactions, medical vocabularies, software design reuse, standards development

FIPA

What is FIPA

- The Foundation for Intelligent Physical Agents (FIPA) is a non-profit association.
- FIPA's purpose is to promote the success of emerging agent-based applications, services and equipment.
- FIPA's goal is pursued by making available in a timely manner, internationally agreed specifications that maximise interoperability across agent-based applications, services and equipment.
- http://www.fipa.org/

Who is FIPA

- FIPA operates through the open international collaboration of member organisations, which are companies and universities active in the agent field.
- Companies: Alcatel, Boeing, British Telecom, Deutsche Telekom, France Telecom, Fujitsu, Hitatchi, HP, IBM, Fujitsu, Hewlett Packard, IBM, Intel, Lucent, NEC, NHK, NTT, Nortel, Siemens, SUN, Telia, Toshiba, etc.
- Universities and Research Institutes: GMD, EPFL, Imperial, IRST, etc.
- Government Agencies: DARPA

FIPA's Work Model

- FIPA's work is built around annual rounds of FIPA specification deliverables.
- FIPA97 laid the groundwork and focused on
  - Agent management (common components, agent lifecycle)
  - Agent communication (message format, semantics, interaction protocols)
  - Agent/Software interaction
- FIPA98 extended fipa97, dealing with
  - Human-agent interaction
  - Agent mobility
  - Agent security
  - Ontology services
- FIPA 99 is work in progress
  - TC1: Agent Management
  - TC2: Agent Communication Language
  - TC3: Agent/Software Interaction
  - TC4-TC7: Specification of Applications

The FIPA ACL

TC2: Agent Communication Language

- Called FIPA ACL
- Based on speech acts
- Messages are actions (communicative actions or CAs)
- Communicative acts are described in both a narrative form and a formal semantics based on modal logic
- Syntax is similar to KQML
- Specification provides a *normative* description of high-level interaction protocols (aka conversations)

Agent-Standardization - FIPA Cooperation between Agents

CAs for Information Exchange

- proposition or reference as content
- Basic CAs:
  - inform
  - query-ref
  - not-understood
- Advanced CAs:
  - inform-if, inform-ref
  - confirm, disconfirm
  - subscribe

Agent-Standardization - FIPA
Cooperation between Agents

CAs for task delegation

- action-description as content
- Basic CAs:
  - request
  - agree
  - refuse
  - failure
  - not-understood
- Advanced CAs:
  - request-when, request-whenever
  - cancel

Agent-Standardization - FIPA
Cooperation between Agents

CAs for Negotiation

- action-description and proposition as content
- Initiating CA
  - cfp
- Negotiating CA
  - propose
- Closing CAs
  - accept-proposal
  - reject-proposal

Agent-Standardization - FIPA
Cooperation between Agents

Example

```
(request
   :sender   (:name user_agent@bond.mchp.siemens.de:3410)
   :receiver (:name hilton_hotel@tcp://hilton.com:5001)
   :ontology fipa-pta
   :language SL
   :protocol fipa-request
   :content
   ( action hilton_hotel@tcp://hilton.com:5001
     ( book-hotel   (:arrival   04/07/1999) (:departure 12/07/1999)
      (:infos  (  ))
     )))
```

FIPA 99: other possibilities to define content!

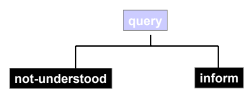
Agent-Standardization - FIPA
Cooperation between Agents

FIPA Cooperation

- CAs have their own formal semantics
  - difficult to implement
  - need not be implemented - agent must behave according to semantics
- Interaction protocols define structured conversations
  - based on CAs
  - basis for dialogues between agents
  - basic set of pre-defined IPs
  - own IPs can be defined

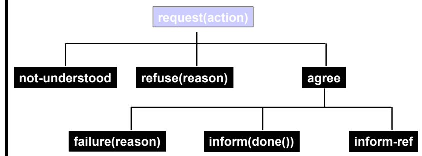
Agent-Standardization - FIPA
Cooperation between Agents

FIPA-Query (simplified - for information exchange)



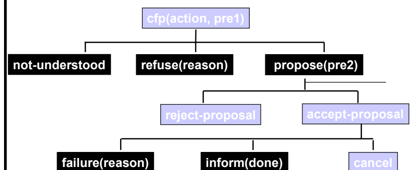
Agent-Standardization - FIPA
Cooperation between Agents

FIPA-Request - for task delegation



Agent-Standardization - FIPA
Cooperation between Agents

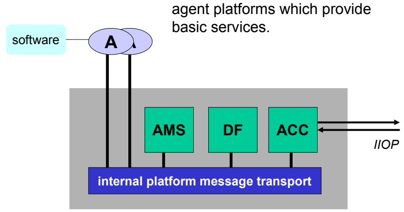
FIPA-Contract Net - for negotiation



The FIPA Agent Platform

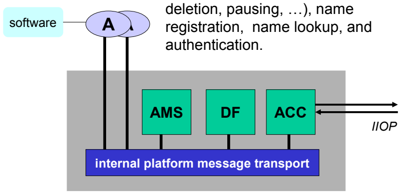
FIPA Agent Platform

Agents belong to one or more agent platforms which provide basic services.



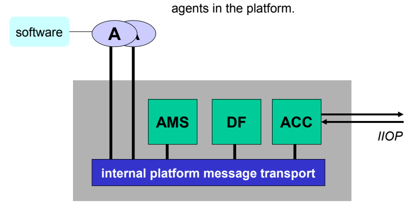
FIPA Agent Platform

The **AMS** (Agent Management System) provides services like lifecycle management (creation, deletion, pausing, …), name registration, name lookup, and authentication.



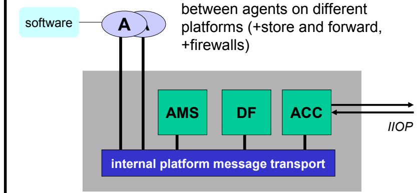
FIPA Agent Platform

The **DF** (Directory Facilitator) provides yellow pages services which describe the attributes and capabilities of agents in the platform.



FIPA Agent Platform

The **ACC** (Agent Communication Channel) accepts and delivers message between agents on different platforms (+store and forward, +firewalls)



FIPA Platform Implementations

- Several platforms have been implemented
  - JADE/LEAP
  - FIPA-OS
  - Zeus (BT)
  - Mecca (Siemens)
  - Spawar
  - Comtec

and interoperability has been demonstrated.

agentcities

- See http://agentcities.com/ http://agentcities.net/
- A network of FIPA platforms
- Each offers a set of services
- Sample services
  - Ping
  - Weather

ACL Semantics

Outline

- Cohen & Levesque
  - Theory of Rational Agency
  - Cohen & Levesque on ACL Semantics
- KQML Semantics (Labrou)
- FIPA ACL Semantics
- Comparing ACL semantics approaches & Comments

Cohen & Levesque

Rational Agency

The Cohen & Levesque Approach

- Most attempts for semantics for ACL descend from the work of Cohen & Levesque (C&L)
- Intention = Choice + Commitment
- Integration of Agent Theory and Semantics of Communication Primitives
- A (partial) theory of rational agency
- Possible-worlds semantics

Commitments and Intentions

- Internal Commitment:
  - (P-GOAL x p q) =
    (1) (BEL x ¬p) ∧
    (2) (GOAL x (LATER p)) ∧
    (3) [KNOW x (PRIOR [(BEL x p) ∨ (BEL x □¬p) ∨ (BEL x □¬q)]
    ¬ [GOAL x (LATER p)])]

  meaning

  "(1) agent x believes p is currently false
  (2) chooses that it be true later
  (3) and x knows that before abandoning that choice, he must either believe it is true, or that it will never be true, or that some q (an escape clause) is false"

Intention

The p of P-GOAL

What has just happened, i.e. x believes that a will happen next and a did happen

- (INTEND x a q) =
  (P-GOAL x [DONE x (BEL x (HAPPENS a))?;a] q)
  - x has the persistent goal of reaching a state at which it believes that a will happen, after which (state) a does happen
- Intending is a special kind of commitment
- The agent is committed to arriving at a state in which he is about to do the intended action next
- Thus an agent cannot be committed to doing something accidentally or unknowingly
- "I intend for the sun to rise tomorrow" vs
  "I intend to get an "A" in this course"

Thoughts on C &L Intention

- Just because an agent intends p, it does not mean that the agent will even attempt to achieve p
  - remember the "escape clause" in the P-GOAL definition a "pessimistic" agent might drop all its goal because "the sky is blue" or for any other reason
- The definition of intention does not guarantee a causal relationship between the agent's action and "an action occurring"
  - the agent is only required to reach a state that the agent believes that will lead to "the action a occurring"

C& L on ACL Semantics

ACLs a la Cohen & Levesque

- C&L object to the use of "performative" to describe KQML's communication primitives
- Communicative acts (CAs) are <u>attempts</u> to communicate
- C&L build on their earlier work on rational agency to define CAs as attempts that involve two (or more) rational agents (teams)
- Interesting work that focuses on defining rational agents and describing team formation.

Semantics for INFORM

Not present in ATTEMPT def'n

The "honest effort"

- {INFORM speaker listener e p} =
  {ATTEMPT speaker listener e
  (know listener p)
  [BMB listener speaker
  (P-GOAL speaker (KNOW listener (KNOW speaker P)))]}
- An INFORM is defined as an attempt in which to make an "honest effort", the speaker is committed to making public that he is committed to the listener's knowing that he (the speaker) knows p.

KQML Semantics

Which Agent States? (Labrou 1996)

- *Preconditions* indicate the necessary state for an agent in order to send a performative and for the receiver to accept it and successfully process it.
- *Postconditions* describe the states of both interlocutors after the successful utterance of a performative (by the sender) and after the receipt and processing (but before a counter utterance) of a message (by the receiver).
- *Preconditions* indicate what can be assumed to be the state of the interlocutors involved in an exchange. Similarly, the *postconditions* are taken to describe the states of the interlocutors assuming the successful performance of the communication primitive

Semantics for TELL

TELL(A,B,X)

- A states to B that A believes X to be true (for A).
- bel(A,X)
- Pre(A): bel(A,X) $\wedge$ know(A,want(B,know(B,S)))
  where S may be bel(B,X) or NOT(bel(B,X))
  Pre(B): intend(B,know(B,S))
- Post(A): know(A,know(B,bel(A,X)))
  Post(B): know(B,bel(A,X))
- Completion: know(B,bel(A,X))
- The completion condition and postconditions hold unless a SORRY or ERROR suggests B's inability to properly acknowledge the TELL.

Semantics for the proactive-TELL

proactive-TELL(A,B,X)

- A states to B that A believes the content to be true.
- bel(A,X)
- Pre(A): bel(A,X)
  Pre(B): NONE
- Post(A): know(A,know(B,bel(A,X)))
  Post(B): know(B,bel(A,X))
- Completion: know(B,bel(A,X))
- The postconditions and completion condition hold unless a SORRY or ERROR suggests B's inability to properly acknowledge the TELL.

FIPA ACL Semantics

TC2: Agent Communication Language

- Called FIPA ACL
- Based on speech acts
- Messages are actions (communicative actions or CAs)
- Communicative acts are described in both a narrative form and a formal semantics based on modal logic
- Syntax is similar to KQML
- Specification provides a *normative* description of high-level interaction protocols (aka conversations)

Outline of FIPA ACL Semantics

- A primitive's meaning is defined in terms of FPs and REs
- The Feasibility Preconditions of a CA define the conditions that ought to be true before an agent may plan to execute the CA
- The Rational Effect is the effect that an agent hopes to bring about by performing an action (but with no guarantee that the effect will be achieved)
- The FPs and the REs involve agents state descriptions that are given in SL

Semantic Language (SL)

- SL is the formal language used to define the semantics of FIPA ACL
- In SL, logical propositions are expressed in a logic of mental attitudes and actions
- The logical framework is a first order modal language with identity (similar to Cohen & Levesque)
- SL provides formalizations for three primitive mental attitudes: Belief, Uncertainty and Choice (or Goal); Intention is defined as a Persistent Goal
- SL can express propositions, objects and actions

An example of FIPA ACL semantics (inform)

$\langle i, \text{inform}(j, \phi)\rangle$

FP: $B_i\phi \wedge \neg B_i(Bif_j\phi \vee Uif_j\phi)$
RE: $B_j\phi$

Agent i informs agent j that (it is true that) it is raining today:

```
(inform
   :sender i
   :receiver j
   :content "weather(today,raining)"
   :language Prolog
   :ontology weather42)
```

Another example of FIPA ACL semantics (request)

$\langle i, \text{request}(j, a)\rangle$

FP: FP(a) [i\j] $\wedge$ B_i Agent(j, a) $\wedge$ $\neg B_i\ I_j$ Done(a)
RE: Done(a)

Agent i requests j to open a file:

```
(request
   :sender i
   :receiver j
   :content "open \"db.txt\" for input"
   :language vb)
```

Evaluation of ACLs and Semantic Approaches

Different ACLs: different semantic approaches

- Different approaches to the semantics of an ACL
  - KQML semantics (Labrou 1996)
  - FIPA ACL (FIPA ACL specification)
  - ACL semantics (Cohen & Levesque)
- KQML's semantics (Labrou 1996)
- Comparison between KQML and FIPA ACL (primarily based on their semantics)
- Cohen & Levesque points on ACLs
- Why not KQML+KIF for an ACL?

Comparison of KQML tell and FIPA ACL inform

- The difference is only observable in the semantics
- Syntactically the two messages are almost identical
- Both languages make the same basic assumption of non-commitment to a content language (in this performative)
- Semantically they differ at two levels:
  - different ways to describe the primitive, i.e., pre-, post-, completion conditions for KQML, FPs and REs for FIPA ACL
  - different language to describe the propositional (mental) attitudes, e.g., KQML's bel is not the same as FIPA ACL B operator

How close can a FIPA ACL primitive get to KQML tell?

$\langle i$, KQML-like-tell(j, $B_i\phi$)$\rangle$

FP: $B_i B_i\phi \wedge \neg B_i(Bif_j B_i\phi \vee Uif_j B_i\phi)$
RE: $B_j B_i\phi$

which can be generated by replacing ϕ by $B_i\phi$ in the definition of inform:

$\langle i$, inform(j, ϕ)$\rangle$

FP: $B_i\phi \wedge \neg B_i(Bif_j\phi \vee Uif_j\phi)$
RE: $B_j\phi$

How do KQML and FIPA ACL differ?

- Different semantics; mapping of KQML performatives to FIPA primitives and vice versa is a futile exercise.
- Different treatment of the "administration primitives"; in FIPA ACL register, unregister, etc., are treated as requests for action with reserved (natural language) meaning
- No "facilitation primitives", e.g., broker, recommend, recruit, etc., in FIPA ACL
- Reserved content language: a very murky issue ...

Does FIPA ACL require a reserved content language?

- The answer is subject to interpretation, but a fair answer would be that YES it does, in some cases
- A distinction has to be drawn between how a message looks (syntax) and what it means (semantics).
- Some FIPA messages (e.g., request) use SL as their content language.
- An agent that observes such messages have to "understand" some SL; how much depends on the particular message

Which ACL should I use?

- Programmers do not care about semantics and their details.
- As long as the agent does not *implement* modalities (belief, intention, etc.) the semantic differences are irrelevant to the developer.
- The similar syntax guarantees that a developer will not have to alter the code that receives, parses and sends messages.
- The code that processes the primitives should change depending on whether the code observes the proper semantics.

Really ... which one is better?

- FIPA ACL is more powerful with composing new primitives.
- The power stems from the power of the SL language as a content language to describe agents' states.
- KQML's weakness is its religious non-commitment to a content language.
- Both have shortcomings; there are features that developers would like to see in an ACL.

Shortcomings of Current ACLs

- Intentional level description: which mental attitudes, what definitions?
- Problems with mental attitudes: from theory to practice
- Can all desirable communication primitives be modeled after speech acts? Should they?
- Flexible description of agents' capabilities and advertising of such capabilities.
- How can we test an agent's compliance with the ACL?
- Ease of extending an ACL

Alternative approaches and languages

Alternatives to ACLs

- There are many alternatives to using ACLs for communicating and sharing information.
- From oldest to newest...
  - Natural language (Espanol)
  - Database languages (SQL, …)
  - Domain dependant (EDI, …)
  - Distributed object systems (CORBA, ...)
  - OKBC
  - Service languages (e-speak, BizTalk, …)
  - P2P and Grid computing
  - Web languages (XML, RDF, DAML+OIL)
- One size won't fit all, so we need to appreciate the strengths and weaknesses.
- We will also see mixing, matching and morphing

NL as an ACL

- Natural languages are the preferred ACL for human agents
- And have strongly influenced the theoretical framework of ACLs for artificial agents
- Some artificial agents accept and interpret NL utterances from humans and also use artificial ACLs to talk to other artificial agents. ("Mr. Data, make it so")
- Some researchers predict a future in which NL utterances will be used as the ultimate ACL for all agents, human and software.

Database Languages

- The database field has developed techniques for sharing information in a distributed environment.
- In fact, it has pioneered the theory and practice of critical concepts like:
  - concurrency control
  - transactions
  - replication
  - security and access control
  - common query languages (e.g., SQL, OQL)
  - common APIs (e.g., ODBC, JDBC)

some of which the agents world has mostly ignored to date.

Domain Dependant Languages

- There have always been specialized languages, protocols, architectures and systems developed for sharing particular knowledge, e.g.:
  - Electronic Data Interchange (EDI) -- designed for sharing well defined business documents (PO, RFQ, …)
  - Z39.50 -- designed to allow an IR client application to talk to an IR backend server.
  - Napster: designed for sharing MP3 files

Domain Dependant Languages Example -- EDI

- EDI involves the application to application exchange of electronic data in support of standard business transactions across enterprise boundaries in such a way that no human interpretation or processing is required.
- Two standards were developed in the 80's: X12 in the US and EDIFACT in the EU.
- These are designed to exchange such business documents as purchase orders, requests for quotations, etc. and their constituent parts.
- Oriented toward integration with legacy systems
- Now being reengineered for XML
- Not very general, expressive, flexible, or extensible

Distributed Objects

- Approaches to sharing objects in a distributed system have been evolving over the past 15 years.
- CORBA
- Distributed Computing Environment (DCE) developed by the Open Group in the early 90's
- Java
  - RMI
  - Enterprise Java Beans (EJB)
  - Jini
- OLE/COM/DCOM/ActiveX (Microsoft)
- SOAP

Distributed Objects -- Typical Components

- A distributed object is an object that can be accessed remotely. An object is typically considered to encapsulate data and behavior.
- Remote procedure/method call
- Interface definition language by which one can specify an object or class's signature, I.e. its methods and the number and types of their arguments.
- ORB (Object request broker)
- Other standard services -- e.g., naming, timing, security, persistence, etc.

CORBA ORB

- Defined by the OMG (Object Management Group) http://omg.org/
- See the CRRBA FAQ for more information http://www.aurora-tech.com/corba-faq/ The ORB is the heart of a CRBA system and mediates communication between clients and servers.
- Inter-orb communication is less standard.
- One can access remote objects by name, or by interface or by capability.
- This last feature is provided by the CORBA trader service

Java

- Virtually all of what CORBA provides is also available in Java via a combination of
  - Java RMI
  - Java RMI servers
  - Beans and enterprise beans
  - Jini
  - Java event servers
  - etc.
- Focusing on a single language has strong advantages and disadvantages.

Java Beans and EJBs

- http://java.sun.com/beans/
- JavaBeans components, or Beans, are reusable software components that can be manipulated visually in a builder tool.
- Typical unifying features that distinguish a Bean are:
  - **Introspection:** enables a builder tool to analyze how a Bean works
  - **Customization:** enables a developer to use an app builder tool to customize the appearance and behavior of a Bean
  - **Events:** enables Beans to communicate and connect together
  - **Properties:** enable developers to customize and program with Beans
  - **Persistence:** enables developers to customize Beans in an app builder, and then retrieve those Beans, with customized features intact, for future use
- Enterprise JavaBeans (EJBs) extends the JavaBeans component model to handle the needs of transactional business applications.

Jini

- Jini provides simple mechanisms which enable devices to plug together to form an impromptu community
- Each device provides services that other devices in the community may use.
  - These devices provide their own interfaces, which ensures reliability and compatibility.
- Jini uses a lookup service with which devices and services register.
- When a device plugs in, it goes through an add-in protocol, called discovery and join-in.
  - The device first locates the lookup service (discovery) and then uploads an object that implements all of its services' interfaces (join).

Jini

- To use a service, a person or a program locates it using the lookup service. The service's object is copied from the lookup service to the requesting device where it will be used.
- The lookup service acts as an temporary intermediary to connect a client looking for a service with that service.
- Once the connection is made, the lookup service is not involved in any of the resulting interactions between that client and that service.
- Jini also defines a leasing and transaction mechanism to provide resilience in a dynamic networked environment.

Javaspaces

- JavaSpaces is a simple unified mechanism for dynamic communication, coordination, and sharing of objects between Java technology-based network resources like clients and servers.
- It is based on the Linda tuple-space model.

http://java.sun.com/products/javaspaces/

Comparison of DCOM/CORBA/Java

ORB	Platform Availability	Applicable to	Mechanism	Implementations
COM/DCOM	Originally PC platforms, but Becoming available on other platforms	"PC-centric" distributed systems architecture	APIs to proprietary system	one
CORBA	Platform-independent and interoperability among platforms	General distributed system architecture	Specification of distributed object technology	many
Java/RMI	wherever Java virtual machine (VM) executes	General distributed system architecture and Web-based Intranets	implementation of distributed object technology	various

SEI ~1997

SOAP: The Simple Object Access Protocol

- SOAP is an XML/HTTP-based protocol for accessing services, objects and servers in a platform-independent manner.
- The current IETF specification:
  http://search.ietf.org/internet-drafts/draft-box-http-soap-01.txt
- The non-binary XML encoding provides flexibility and platform independence
- The HTTP transport obviates many firewall problems.
- Offers several basic message oriented protocols, including *request-response* and *fire-and-forget*.
- BizTalk builds on SOAP, adding more service oriented features, such as QOS and routing information

SOAP: The Simple Object Access Protocol

An example of a SOAP request-response pair

A SOAP Request

```
<SerializedStream SerializationPattern=
 "urn:schemas-microsoft-com: soap:v1"
 headers="#ref-0" main="#ref-1">
 <headers id="ref-0">
   <InterfaceName>

soap:cdl:com.develop.demos.purchase_b
ook
   </InterfaceName>
   <CustomerID>
    CDFE06E0-4DB4-4809-A7CF-
4DDA32D5B081
   </CustomerID>
 </headers>
 <PurchaseBook id="ref-1">
   <_ _param0>0201379368</_ _param0>
 </PurchaseBook>
</SerializedStream>
```

The SOAP Response

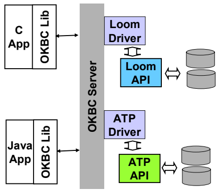
```
<SerializedStream SerializationPattern=
 "urn:schemas-microsoft-com:soap:v1"
 headers="#ref-0" main="#ref-1">
 <headers id="ref-0">
   <InterfaceName>

soap:cdl:com.develop.demos.purchase_boo
k
   </InterfaceName>
   <CustomerID>
    CDFE06E0-4DB4-4809-A7CF-
4DDA32D5B081
   </CustomerID>
 </headers>
 <PurchaseBook id="ref-1">
   <_ _param0>0201379368</_ _param0>
 </PurchaseBook>
</SerializedStream>
```

Open Knowledge Base Connectivity

- http://ai.sri.com/~okbc/
- OKBC is to KBs what ODBC is to Databases -- defines a standard API for frame-based KR systems
  - Provides two access protocols (frame oriented operations and a sentential tell/ask) and a linear batch language (def-okbc)
  - Supports a client-server model for interaction
  - Provides an object-oriented view of a KRS
  - Supports wide variation in underlying KRS
- Adopted as KRS interoperation protocol within DARPA High Performance Knowledge Base (HPKB) program
  - OKBC drivers available for Loom, Ontolingua, Ocelot, ATP, JavaKB, TupleKB, ...
  - OKBC applications include GKB (SRI), Jot (KSL), Ontolingua (KSL), Riboweb (SMI), Protégé (SMI), Hike (SAIC), …

OKBC Runtime Architecture

- Client-server architecture
- Client libraries in Java, C, Lisp
  - From single source for consistency
- Servers in Java, Lisp
- Drivers for Loom, Ontolingua, Ocelot, ATP, Snark, Cyc, …
- Transparent network access
- Extensible connection model
  - Allows alternative security, authentication policies
- Support for efficient networking
  - Client side caching
  - Remote procedure language
  - Enumerators with prefetch

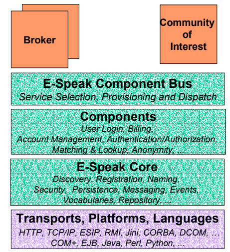


Service Languages

- By service languages we mean a new class of languages designed to facilitate service description, service discovery, and application data exchange.
- Examples include:
  - toolTalk (DEC circa '96)
  - e-speak (HP)
  - BizTalk (Microsoft et. al.)

HP E-Speak

- See Http://e-speak.org/
- Defines a language and protocols at the service layer
- Initial software components released under GPL
- **Services:** *Broker, Discovery, Mediation, Composition*
- **Specification:** *Contracts, Vocabularies, Services*
- **Security:** *E-Speak core as a "sandbox", local names, certificates*
- **XML, Java:** *queries, vocabularies, messaging*



Microsoft's BizTalk

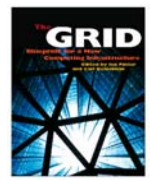
- See http://biztalk.org/
- BizTalk is an industry initiative started by Microsoft and supported by a wide range of organizations, from technology vendors like SAP and CommerceOne to technology users like Boeing and BP/Amoco.
- BizTalk Framework™ is a set of guidelines for how to publish schemas in XML and how to use XML messages to easily integrate software programs together in order to build rich new solutions.

Web Languages

- As the web becomes increasingly pervasive and important, it's specialized languages for representing and sharing information are becoming more significant.
- Some key web languages
  - HTML and web scraping
  - XML and associated DTDs
  - RDF and associated standards (e.g., RSS, PICS)
  - More expressive web languages, such as SHOE and DAML

Grid Computing

- See http://www.gridforum.org/
- Flexible, secure, coordinated resource sharing among dynamic collections of individuals, institutions, and resource
- Enable communities ("virtual organizations") to share geographically distributed resources as they pursue common goals -- *assuming the absence* of … central location, central control, omniscience, or existing trust relationships.
- Inspired by early systems like SETI



P2P

- P2P systems started out as simple file sharing systems like Napster
  - One or several simple data types
  - Well defined data types
- Being generalized to more data types with beter meta data
- Moving from centralized servers which act as registrars and meta data repositories (e.g., Napster) to fully decentralized systems (e.g., Gnutella)

Conclusions

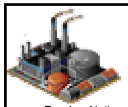
- One size won't fit all
- General purpose vs. specialized languages
- Things will continue to evolve
- Advice: if you are implementing a single application, before you go with an ACL, convince yourself that it's not right for either (1) a database approach or (2) a distributed objects approach

Applications

Examples of Applications using ACLs

There have been a number of large R&D applications which used ACLs in an integral way.

- **Carrot** - distributed information retrieval
- **CIIMPLEX** - Manufacturing planning and scheduling
- **Kimsac** - Advising on government services
- **Unisys NLA -** Speech system toolkit
- **UMDL -** Univ. of Michigan Digital Libraries
- **Infomaster -** Information integration



CIIMPLEX EECOMS

Consortium for Integrated Intelligent Manufacturing Planning and Execution

Extended Enterprise Coalition for Integrated Collaborative Manufacturing Systems

- **Funder:** National Institute of Standards and Technology / Advanced Technology Program
  - Technologies for the Integration of Manufacturing Applications (TIMA)
  - ~ $45M over six years in two ATP projects
- **Goal:** Plug and Play framework of business objectives and integration-enabling tools allowing a suite of solutions that can be implemented "out-of-the-box" at small and midsized manufacturing and process sites including MES, ERP, Finite Scheduling, and Capacity Analysis/Decision Support
- **Objectives:** interoperability, configurability, adaptability, extensibility, plug and play.

Participants

- IBM Corp
- Universities
  - University of Maryland Baltimore County
  - University of North Carolina at Charlotte
  - University of Florida
- Berclain USA Ltd.
- Boeing
- QAD Inc
- GSE Systems
- Lucent Technologies
- Ingersoll-Rand Co.
  - Demand Solutions
  - DLoG Remex Inc.
- Intercim
- EnvisionIt Software
- The Haley Corporation

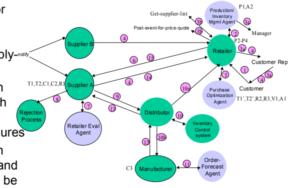
Manufacturing Enterprise Integration

- Integration of planning and execution is imperative for agile manufacturing
  - parts delivery is delayed by the part supplier
  - a preferred customer asks to move ahead a delivery
  - machine breaks down on shop floor
- This involves collaboration among business applications and managers
- Business applications are legacy systems
  - not intended to talk to each other (no API, no means of communication)
  - developed over long period of time (expensive to change)
  - many decision steps are not covered (white space between applications)
- Multi-agent system (MAS) approach
  - flexible and dynamic communication among applications
  - plug-and-play
  - interface agents to interact with people
  - other agents to fill the white space between business applications

Negotiation among agents in the Supply Chain

Goals

- Support automated or semi-automated negotiation among applications in a supply-chain
- Develop an approach that can integrate with existing business practices and procedures
- Develop an approach that uses standards and technologies likely to be acceptable to the business community



Specific Approach

1 Use (modified) FIPA ACL primitives for negotiation
  – Important contribution is the set of primitives and their semantics

2 Use XML, extended with KIF, as the content language
  – KIF-based extensions allow the use of constraints and business rules

3 Introduce the notion of adjustable autonomy into agent-based supply chain negotiation
  – Use of *"decision rules"* to decide how to respond augmented
  – with *"authorization rules"* which decide if the action should be reviewed for authorization and by whom.

Negotiation primitives

- Based on the FIPA ACL with extensions
- Basic negotiation primitives:
  - **cfp**: call for proposals
  - **propose:** propose (or counter-propose) an action
  - **accept-proposal:** accept a proposal
  - **reject-proposal:** reject a proposal (with optional reason)
- Other ACL primitives useful in negotiation
  - inform, query, request, not_understood, refuse, ...
  - advertise, subscribe, broker, register, …
- Specific negotiation protocols are defined using these primitives
  - e.g., Iterated-contract-net, English-auction, etc.

Defining negotiation protocols

- Different protocols can be defined using the communicative primitives
  - contract-net
  - iterated contract-net
  - English auction
  - etc
- Most protocols can be defined with a simple deterministic finite-state automata (DFA) formalism. More complicated ones will require CPNs.
- Negotiations can be augmented by "side conversations" composed of queries, informs, etc.