Lecture 4:
Putting coordination in a wider perspective

The role of coordination in network-aware applications

An exercise in multiagent coordination

Designing WWW-oriented multiagent applications

Active documents are agents

Coordination architectures for document-agents

Issues in agent-oriented software engineering

Acknowledgements: most ideas exposed here were developed in discussions and joint papers with my colleagues and students in Bologna, especially Andrea Omicini, Fabio Vitali, Franco Zambonelli, Luca Bompani, and Davide Rossi.

An exercise in exploiting coordination

Task: Redesigning a program as a multiagent system

Coordination languages like Linda were born as tools for parallel programming environments on high performance architectures, but they can also be effective on networks

The "computer is the network": coordination languages enhanced their importance as tools for designing network-aware applications, where interaction control is a must

The Tuple Space coordination model is especially effective as a basis to design multiagent applications

Multiagent system:
⇒system with several "autonomous" components
⇒system with several "logically mobile" components

Redesigning a game-playing program as a multiagent system

We have studied game-playing programs (chess) implemented over a network of workstations, aiming at exploiting the combined computing power of several workstations

The lack of a basic theory of distributed search is the main problem in parallel game playing; experiments are useful, but often difficult to evaluate wrt other approaches: different sw, different hw, different programming styles

We have developed a number of experiments, aiming at systematically testing the effectiveness of two different software architectures implemented with the same software and running on the same hardware

Tools

- Network C-Linda on SUN workstations
- GnuChess, a chess playing program written in C

Results:

- Comparison of parallel search algorithms in Linda
- A new cooperation model for game playing: multiagent system based on distributed knowledge

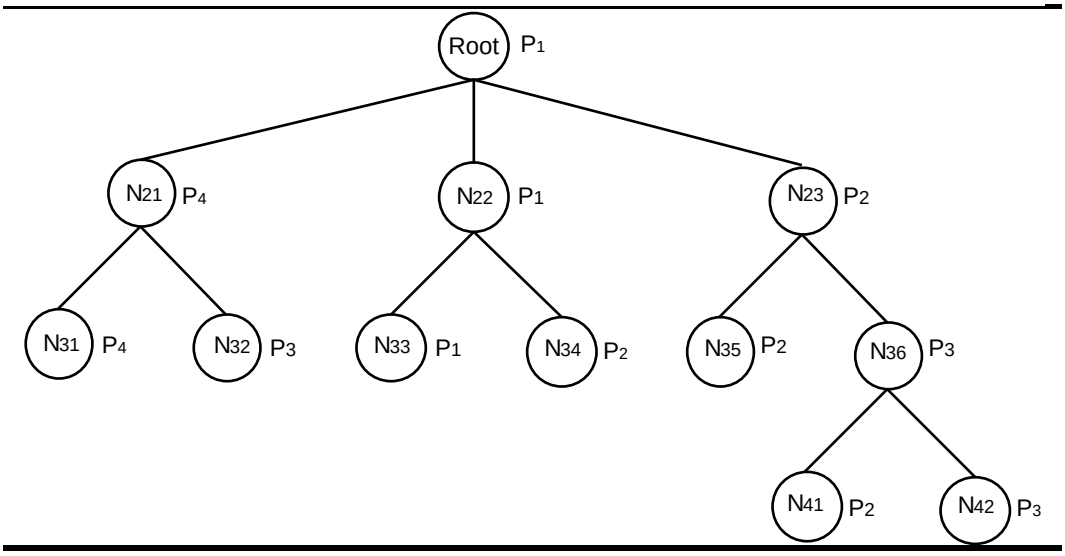
Parallel search of game trees

i) Parallel aspiration search [Bau78]: the alphabeta window is partitioned in a number of contiguous segments, that are used by different processors to explore the same game tree; this is a form of OR-parallelism.

ii) Parallel evaluation by special hardware; *e.g.*, HITECH uses parallel hardware for move generation [Ebe86], while Deep Thought [Hsu90] uses parallel hardware for evaluating positions.

iii) Mapping the search space on the processor set by a hash function: this is advantageous if the search space contains several duplicate states; *e.g.*, [Sti91] describes how the whole search space of some chess endings was mapped on a Connection Machine.

iv) Parallel search of "splitted" game tree: the game tree is *decomposed* assigning its nodes to different search agents



Naive parallel alphabeta in Linda

A decomposition algorithm (naively) written in Linda

```
/* each search agent knows two main values:
ME= agent identifier; NSONS= # subprocesses */

#define update_alpha(VALUE)
{if (VALUE>alpha) alpha=VALUE;
if (alpha>=beta) /* all subprocesses terminate */
  {terminate();return(alpha);}
}

int treesplit(position p,int alpha,int beta,int depth)
{
int nmoves,result,free,ind;
position *successor;
if (depth==0) return(evaluate(p));
if (NSONS==0) return(alphabeta(p,alpha,beta,depth));
nmoves=genmoves(p,&successor);
if (nmoves==0) return(evaluate(p));
for (i=0,free=NSONS;i<nmoves;ind++)
  {eval("split",ME,treesplit(*(successor+ind),
                             -beta,-alpha,depth-1));
  free--;
  if (free==0)
     {in ("split",ME,?result);
     free++;
     update_alpha(-result);}
  }
while (free<NSONS)
  {in ("split",ME,?result);
  free++;
  update_alpha(-result);}
return(alpha);
}
```

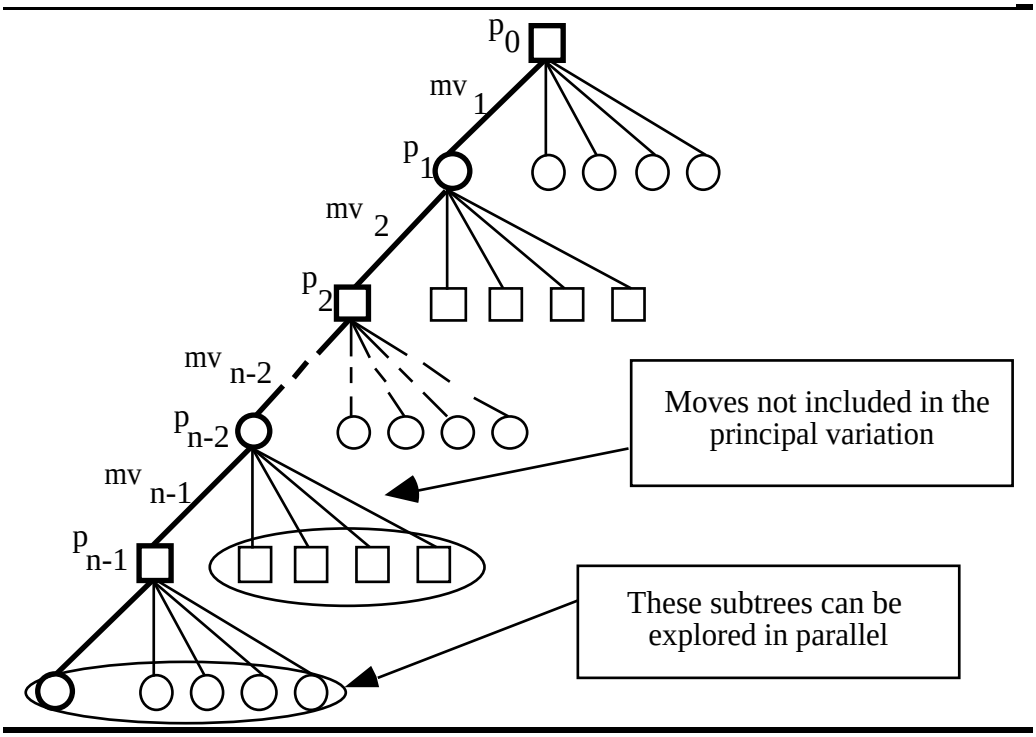
Game tree splitting

Most game tree search algorithms cannot be implemented efficiently over a network, because of process generation and interprocess communication overhead

In order to limit such overhead the concept of tree splitting has been introduced; parallel search algorithms can then be classified in two classes:

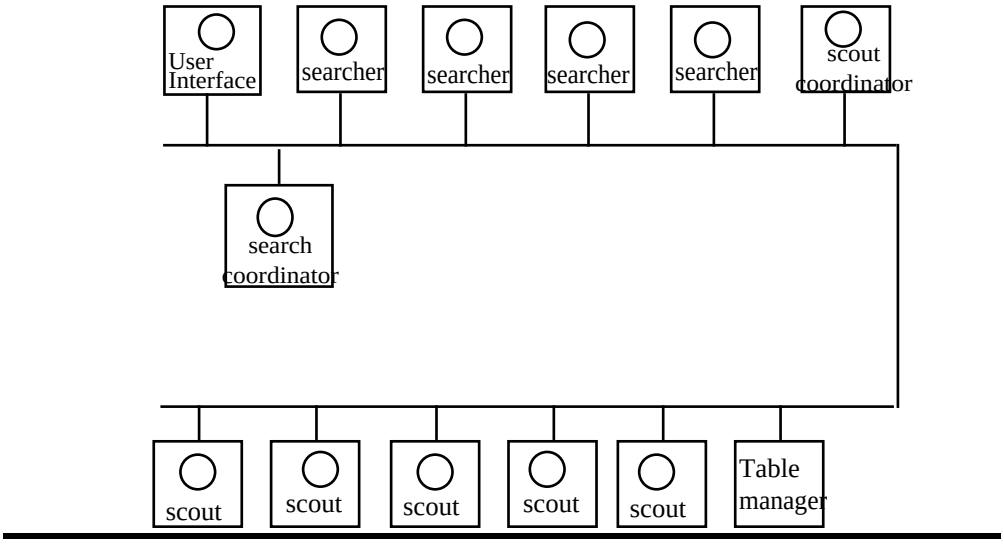
• static splitting: the game tree is decomposed and assigned to processors before search starts

• dynamic splitting: each node in the game tree can be a splitting node - the decision is taken at runtime



Knowledge distribution and cordination

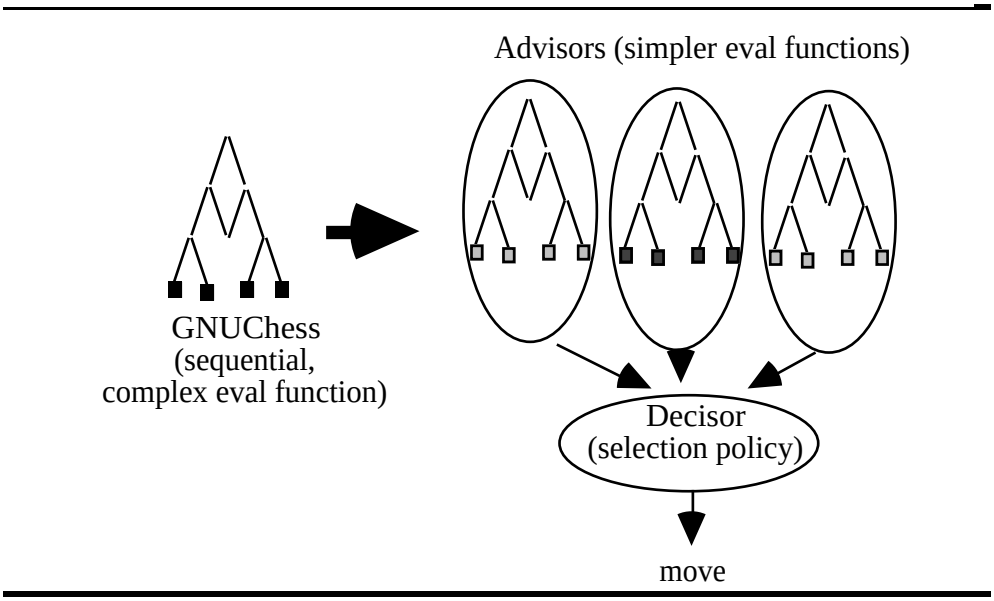
Schaeffer (1987) showed that no speedup is gained with more than 10 processors over a network



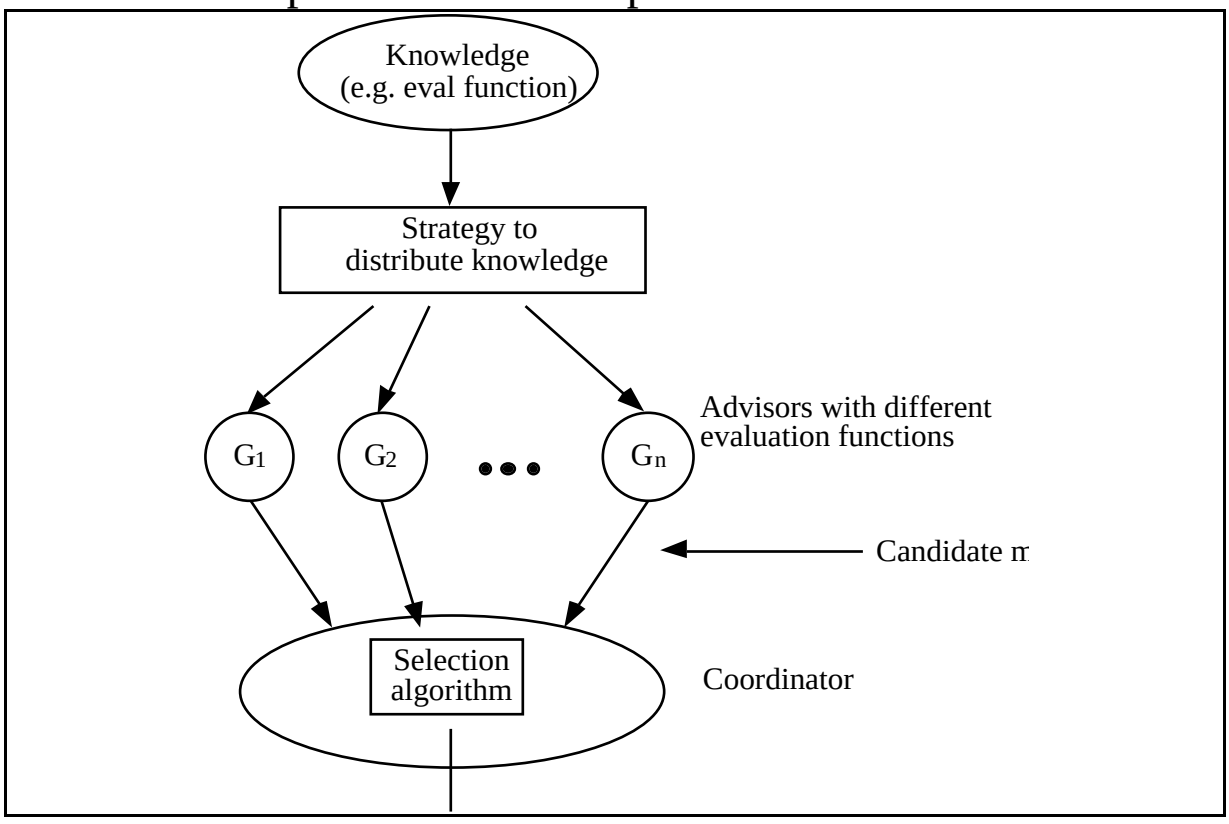
ParaPhoenix (a chess program distributed over a network):
• scouts: no knowledge, high speed to explore the game tree more in depth only checking for material losses
• searchers: “normal” workers able to explore a splitted tree

Althofer (1991) made a number of experiments in which a human (himself) cooperated with two different machines: he found an improvement of 200 Elo points over its own rating

A multi-agent approach

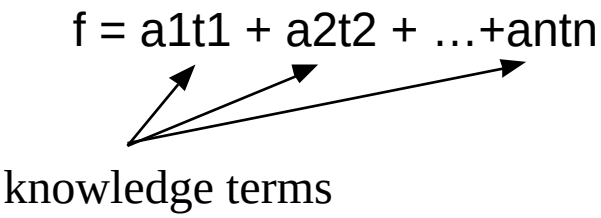


How knowledge should be distributed?
which selection politics should be applied?
how selection politics can be improved?



Distributing knowledge

We distribute "terminal knowledge", i.e. knowledge used to evaluate game tree leaves, usually embedded into polinomials

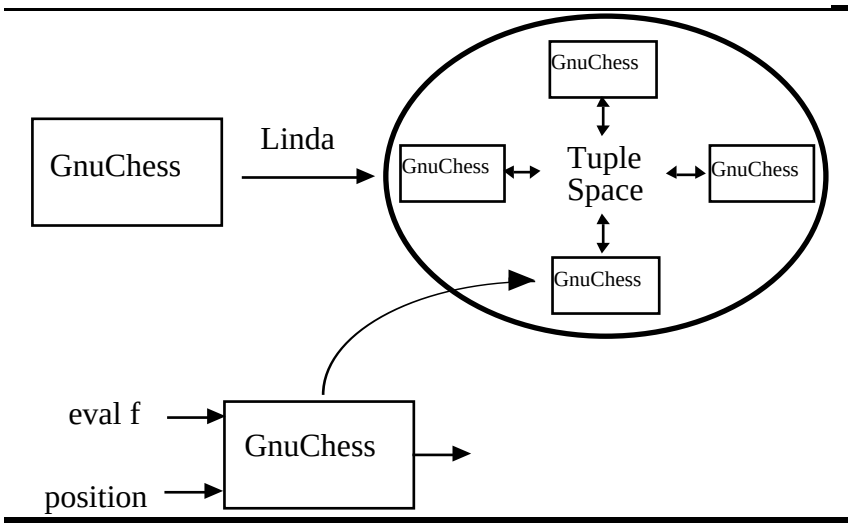


The GNUChess evaluation function contains the following terms:

f(position) =

material (M) +
mobility (X) +
positional value of pieces (B) +
king safeness (K) +
control of center (C) +
pawn structure (P)+
attack/piece protection (A) +
relationship pieces/pawns' structure (R).

We used GNUchess own evaluation function (containing 8 knowledge terms) to obtain a number of "advisors"



Results

On distributed search:

• we have formalized the approach of game tree splitting
• we have confirmed perfomances of classic algorithms in a unified hw/sw environment
• we have developed a general algorithm for dynamic splitting written in Linda

On knowledge distribution:

• we have defined the new approach
• we limited ourselves to terminal knowledge
• we explored a number of distribution and selection criteria
• we applied genetic algorithms to improve the selection

We need theories for both parallel search of game trees and distribution of knowledge among cooperating agents!

The Internet as a programmable platform

"The computer is the network": the convergence of Information and Communication Technologies produces new opportunities for industry, research, and teaching.

Example 1 The mp3 format and the Napster service are challenging the music industry: the old ways of distributing, selling, storing and playing music are obsolete

Example 2 Several Universities are now offering courses for designers, managers, and even art directors of WWW-based enterprises: we started forming people for the "contents industry" over the Internet

The convergence, or "networking", of ICT industries is pushing the development of novel appliances, applications, services, and even organizational theories where the Internet plays a major role

Computer scientists and sw engineers are challenged to adapt themselves to the new platform, inventing new methods to expoit the new computing models enabled by the Internet

Remark:
we should probably start speaking of "Internet science" rather than "Computer science"

The Internet as a platform for groupware

An important application of the Internet is in the field of *groupware*, that is a domain offering interesting and important design problems

Groupware: multiuser, document-centric applications

- *mobility* of people, hosts, and documents
- *communication*: synchronous (“same time”) or asynchronous (“any time”)
- multi-user *interaction*: “same place” or “different places”
- *composition*: groupware is usually the result of the combination of several software technologies
- *agenthood*: in groupware several activies can be performed by autonomous programs (that are possibly mobile)
- *document-centric*: documents are complex data structures with user-specific contents, structure, and behaviours

WWW-based, agent oriented groupware

The software industry is redefining itself into a sw-intensive service industry: eg. Microsoft says that its main competitor is no more the software producer Oracle, but the Internet Service Provider America On Line.

Currently most sw-intensive services are actually Web applications supporting some form of groupware

Example: Microsoft Hailstorm

WWW-based, agent-oriented groupware is quite challenging:

⇒ software technologies related to the WWW are fastly evolving, usually as a result of some standardization process (by organizations like W3C or OMG)
⇒ classic software engineering techniques are unsuitable for Internet applications: in fact, we need novel network-aware agent-oriented ontologies and tools
⇒ these applications usually offer services to different organizations of the real world, which usually use different ontologies for documents, services, and agents

Mobile entities in the WWW

The original WWW was based on two separate concepts:

⇒HTTP servers distribute documents on demand to client browsers: this is *mobility of data* (HTML and XML are not Turing equivalent: they are just SGML dialects to specify data structures like paragraphs or tables)

⇒A browser can "navigate" through the hypertext links, requesting HTML pages to a server, and sometimes "jumping" from a server to another server. Although URLs denote static, "physical" resources, they can be passed around: this is *mobility of references*

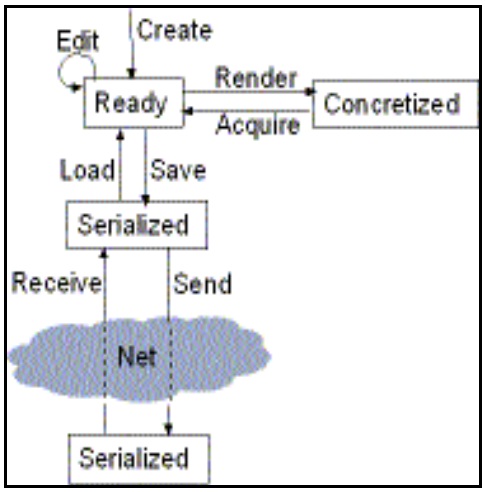
Remark: Mobility of reference means both that a channel name can be passed around, and that a process can detach a channel and connect to another channel

A third concept of mobile entity:

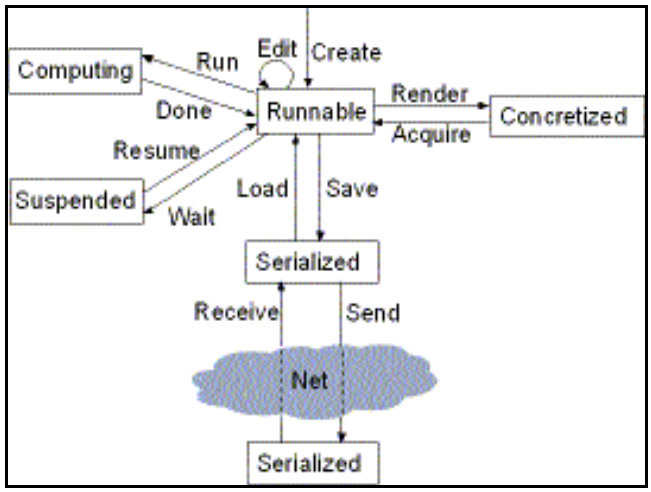
⇒An **active document** (= contents+structure+behavior) moves across the net, from servers to browsers and back

Technologies for active documents

Document lifecycles:



Passive document



Active document

An “active” document provides support for *interactions*; that is, it can include animations, perform computations, provide support for searching, etc.

Active-X objects, Java applets, JavaScript scripts or even complex PostScript or TeX programs can be used to build documents offering more than their content

The idea of “active document” consists of putting the behaviour together with the contents: using **generic markup** behaviors and actions are applied just like formatting

Remark: an active document is an (autonomous) entity including code and data, and moving around: it is an agent!

Representing documents

We use the term “document” with a broad scope, meaning any kind of data structure which network-aware applications can exchange

(passive) document: contents + (structured) representation

Example: RTF is a language of commands that a word processor has to apply to a document to render its contents, in terms of fonts, justification, margins, etc.

HTML has been invented to write passive documents to be displayed inside WWW browsers

(HTML) document: contents + procedural markup

Document processing applications use two different approaches to represent a document

a) a proprietary, binary, *machine-readable*, code:
   Examples: MS Word, Adobe PDF, SUN Java bytecode
b) an open (standard), ASCII-based, *human-readable* code:
   Examples: RTF, HTML, PostScript, TeX

Remark: ASCII-based documents are “flat”: their structure either is “wired” inside the applications which first parse then can manage them, or some further code (markup) is needed to give structure to an ASCII file

Procedural vs declarative markup

Formatters (eg. TeX or PostScript) assign a rendering behaviour to documents represented as files mixing formatting commands (*mark-ups*) and text

In order to build a “page”, formatters are driven by markup commands interspersed in the document text: formatters are in fact compilers

A system such as TeX produces high quality results because layout algorithms are able to approximate the behaviour of experienced professionals

However, TeX (and HTML) markups are very procedural, and this is bad for two reasons:

- the logical structure of a document is not expressed in the markup, thus searching document abstractions (eg. all titles, represented by indented bold lines) is difficult
- the concept of style is non existent, thus changes in style require revising all markup commands

A solution was the introduction of declarative markup languages (like LaTeX), useful to declare both the logical structure of a document and the styles to be used

Example

⇒`\documentstyle[twocolumn]{article}`

Towards active documents: XML

XML is a standard markup language (defined by W3C, and derived from SGML) to describe the *structure* of documents; instead, documents *behaviours* are specified using either stylesheet languages (e.g. XSL) or even programming languages (e.g. Java)

Remark: declarative markup is either structural or semantic

Structure: A book is composed of chapters, sections, titles, notes, etc. A letter is composed of sender, addressee, salutations, body, signature, attachments, etc.

Semantics: A news item about a criminal act may specify the source of the news item itself, the description of a sequence of acts, the name of the place where the acts took place, the name and rank of the involved police officers, the stolen amount, etc.

Per se, XML is just a language to describe tree-like data structures; its companion languages, especially XSL-T, are able to deal with semantics

document: contents + structure + behaviours

What is a hypertext document

Definition: a *hypertext document* is defined by

- its contents,
- its structure,
- one or more “behaviours”, and
- its relationships with other documents

Intuitively, a document carries some *information* and has some *structure*: a book, a report, a letter, a program are examples of documents of different forms

When documents live inside a computer they have a *physical representation* based on some data structure

When we consider a document in abstract, it is fully defined by its contents and logical structure: the structure of a document is an instance of a *document model*, that is an ontology of abstract entities suitable to describe the document’s elements (eg. chapters, sections, paragraphs, pages, etc.)

Any document can display several *behaviours*:

⇒ a rendering, or presentation behaviour, defines how a document is displayed on an output device, like a screen or a printer (e.g. pretty printing is a rendering behaviour)

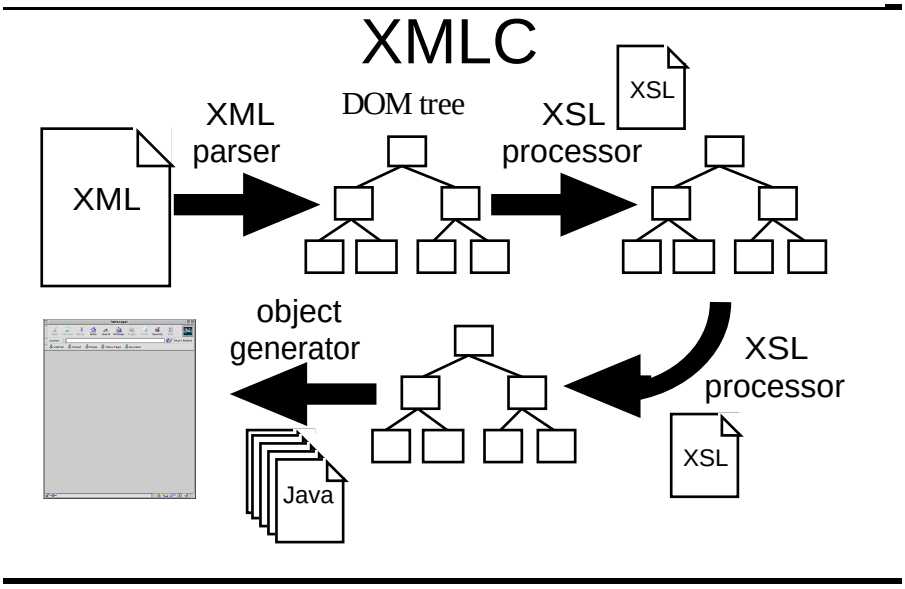
⇒ a view, or control behaviour, defines how a user can interact with a document (e.g. using hypertext)

⇒ a static semantics defines how a document can be analysed with respect to some verification rules

Active document are agents

XSL is not up to XML in terms of generality: specialized notations are not supported. At UniBologna, we have created an open set of formatting objects that are loaded when needed, depending on the required behavior of a document.

Each document has a stylesheet associated that maps its elements to the available formatting objects; each formatting object is then associated to a sw module (a JavaBean), that we call *displet*, displaying the information (we exploit the dynamic linking capabilities of Java)



Some applications that we have explored:

- Special typographical elements; layout management
- Notations for software engineering documents
- Management system for hypertext UML documents
- Porting of ToolBooks inside standard browsers
- DSS for financial applications
- WorkSpaces (a workflow management system)

Declaratively active documents

We have two reference architectures for displets, that we call “server-side” and “client-side”, respectively

The *server-side* architecture is more efficient, but less flexible (a behaviour is pre-processed and “wired-in” a document, that then is sent to a browser and displayed)

The *client-side* architecture is a multiagent system that can be used to have documents performing (coordinable) activities, rather than just be displayed

Since we associate displets which are Java Beans to XML documents, we can ask a Bean to paint itself, or to perform any other method of its classes

Depending on the stylesheet and the Java classes, then the same document can behave in any of different ways

The code performing the activities is not part of the document (as in Active X or similar systems) but declaratively associated to the document

Example: Music scoresheets
We have defined a DTD for music scoresheets, thus we can represent textually any music score. Then we have defined stylesheets to display, animate, and play a score. Then we have enriched the displets able to display scores with editing capabilities, so that the original document can be modified

The impact of agent-hood

The WWW made clear that documents should be considered as portable, application-independent components requiring specific models and languages

However, the design of WWW-based groupware needs more than just a technology for sw components: documents are not only active; they are *interactive* agents (their users play some role) and moreover they activities need to be coordinated

Databases were the focus of application design in the ’80
Components were the focus of application design in the ’90
Agents are being the focus of application design in the ’00

Component-based applications are usually built on top of a distributed middleware platform

⇒ CORBA (Common Object Request Broker Architecture)
⇒ Lotus Notes
⇒ World Wide Web
⇒ Sun’s Jini
⇒ Microsoft .NET initiative

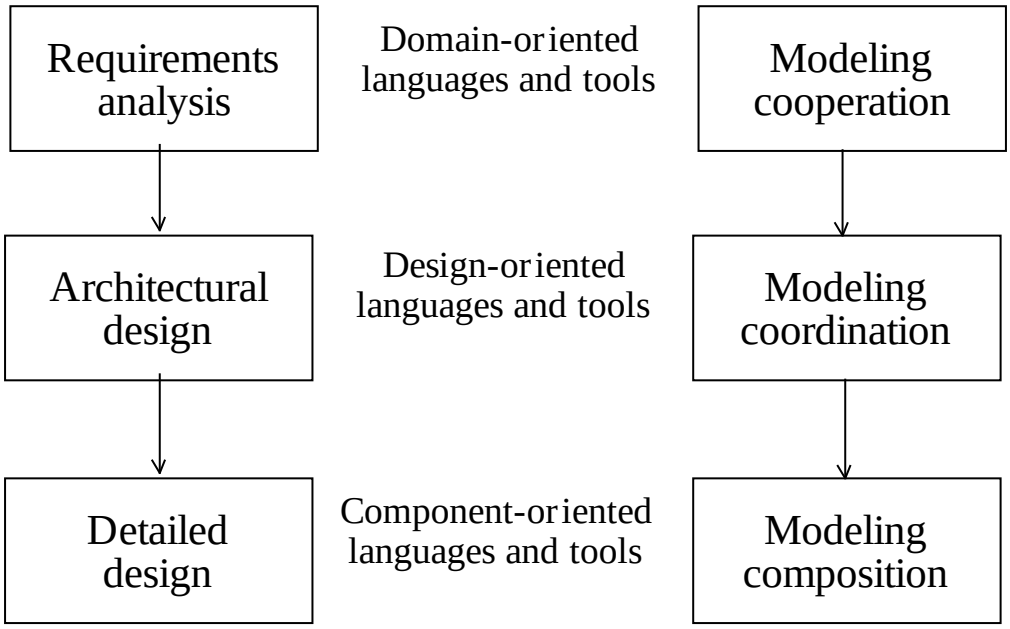
All these architectures offer some concept of *agent-hood*

Agent oriented sw engineering

Sw engineering deals with development processes and tools

"Classic" sw engineering is not adequate for the network, because it lacks of conceptual tools to deal with *interaction*

Classic vs. agent-oriented sw engineering process:



⇒ *Cooperation model*: the abstract description and analysis of all roles involved in a system (cf. Use cases in UML)

⇒ *Coordination model*: the description of a sw architecture in terms of agents and their activities (eg. UML is weak in dealing with agent-oriented architectures)

⇒ *Composition model*: the set of mechanisms used to implement, reuse, or activate components and assign resources (eg. "JavaBeans components representing XML documents and roaming a web of HTTP servers")

A case study in Internet groupware

International scientific conferences are organised involving several people distributed all over the world
- authors of papers
- reviewers of papers
- Program Committee members
- Program Chairs (co-ordinators)
- conference organisers

The following activities have to be performed:
0. The "social laws" of conference management are stated
1. PC-members state their competence fields
2. Authors submit their papers by a given deadline
3. PC-chairs distribute papers according to the competence
4. Reviewers return they evaluations by a given deadline
5. PC-members decide according to the social laws
6. Authors of accepted papers prepare a final version
7. PC-chairs edit the proceedings
8. Organisers handle the list of talking/attending people

⇒Most people involved are mobile: scientists travel a lot
⇒Activities involve people cooperating on structured docs
⇒Most activities are routinary and can be managed by automatic scripts exploiting e-mail/ftp/WWW services

How should we design applications like a conference management system in terms of cooperation laws, coordination mechanisms, and component architectures?

Conference management over the Internet

Basic idea: submitted papers are agents (active documents)

Cooperation model: *which social laws for roles and activity workflows?* we have to decide which conference organization we prefer:

single/multiple tracks and conference workflow;
tyrannical, oligarchic, democratic cooperation management;
authors of papers anonymous/known to reviewers; etc.

Coordination model: *how do we organise agents?*
we have to decide which sw architecture we offer to the agents, eg. a database-like or a peer-to-peer or a MUD-like one, and how agents interact with their environment

Composition model: *how do we reuse and compose components into architectures?*
we have to detail which components we allow and how do they "connect", eg. we could ask that all papers are based on PostScript (so that some browser could manage them) or on XML/XLink-Xpointer (so that they could form an hypertext and be managed by a search engine) or on .NET components

The impact of software architectures

The importance of studying software architectures and their related co-ordination models cannot be overestimated:

Napster is a service based on a peer-to-peer software architecture: its success is so big that it is redefining the music industry, and inspiring novel services and projects (eg. Sun's Peer-to Peer networking initiative)

In order to build a new generation of Internet-aware programming languages, we need to study and understand how co-ordination can be modelled and embodied in a software architecture

Agents, societies, organizations

Internet-aware applications are designed by integrating several technologies; from a sw engineering perspective, we need methods and tools to master the complexity of such an integration

In my opinion the most interesting questions are open to be studied in the field of modeling, specifying, and analysing the *organisations* of agents. For instance, we need notations to describe and study e-commerce service chains, or logistic support systems

The increasing success of the concept of ERP practices and support systems shows that the introduction, or the adaptation, of Internet-aware services to a company reshapes its organisational forms and behaviours

We lack of notational and reasoning tools able to support the analysis of organizations, be them made of societies of agents or chains of services

Research on agent-based software is currently spread over several computer science sub-disciplines, like for instance Artificial Intelligence, Distributed System Programming, Network-aware Programming Languages, Information Retrieval Systems (a very partial list!)

A task for Agent-Oriented Software Engineering researchers: to develop a uniform framework including specific development process models, meta-models for co-operation, co-ordination, and composition, and tools and environments to support all the related design activities

Communication, cooperation, coordination

	synchronous	**asynchronous**
communication	Messaging, chat	e-mail
cooperation	Napster	ftp, HTTP, WebDAV
coordination	distributed game playing (MUD), auction system	workflow (eg. conference management)

Communication mechanisms allow to exchange msgs and/or data streams; these are the basic services needed to build applications like IRC (Internet Relay Chat), e-mail, or teleconferencing systems

Cooperation mechanisms allow to share documents and resources; groupware applications like Napster and shared data repositories need specific cooperation protocols, respectively synchronous or asynchronous.

Coordination mechanisms allow the orchestration of multiple activities and services; sw platforms based on some coordination architecture are useful to build distributed game-playing environments or workflow support systems

Applications requiring coordination management: examples

Example 1:
Parallel simulation. Simulation models based on some notion of discrete, backtrackable time, require specific coordination techniques for a parallel implementation

Example 2:
Integration of whole applications, reconfiguring and coordinating the computations of several independent decision support systems (eg. inter-related spreadsheets under the control of different users)

Example 3:
Multiagent symbolic computing, as in an environment integrating theorem proving with model checking and other reasoning tools

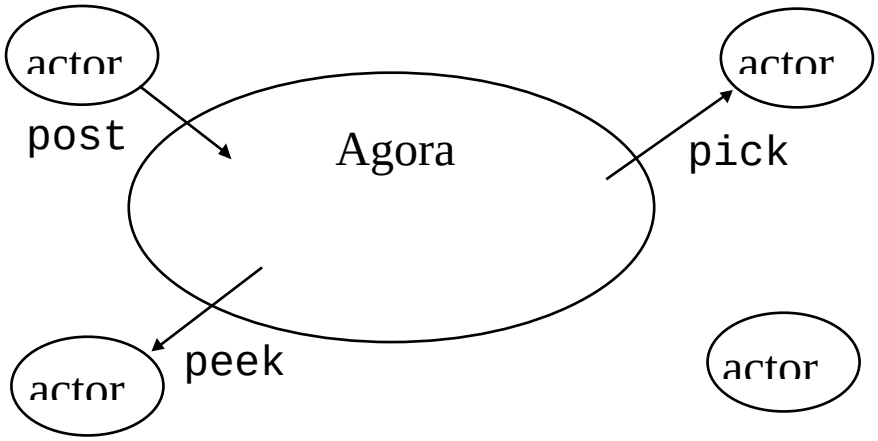
Example 4:
Workflow systems for active documents. Modern document management systems are usually based on a notion of document-agent whose interactions with users, tools, and other documents-agents have to be explicitly managed

Workflows in Sonia

Sonia [Banville 96] is an extension of Linda to support workflows in organizations

Sonia includes the operations post (Linda'sout), pick (in), peek (rd) and cancel (abort a peek or pick); both post and pick can have a timeout (like in Objective Linda)

Tuple matching laws are:
- by value
- by type
- any (polimorphism)
- by rule written in Smalltalk



Software process workflow

The development of a software product can be described by a *software process*

Example: a very simple software process

edit a (single module) program, then compile it, then if compile-ok then execute it

Software processes should be explicitly represented by formal languages [Osterweil 87]

Some goals:

• the software process states the "laws of interaction" among agents involved in software development

• the software process model helps in estimating and planning needed resources

• process descriptions can be reused, studied, measured

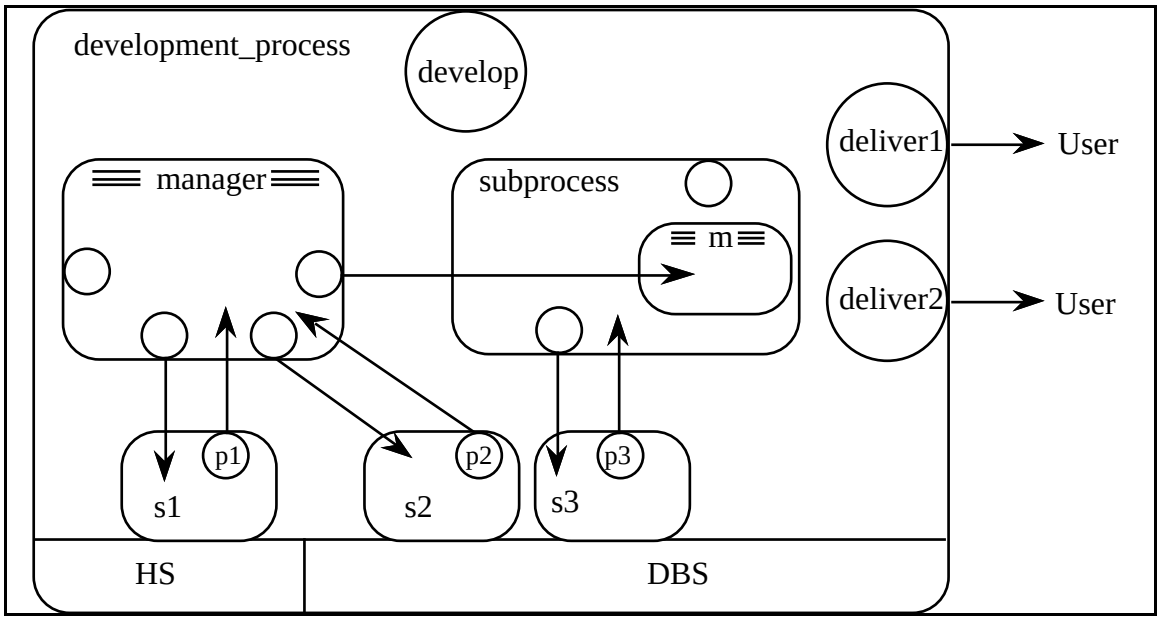
• formal process descriptions can be animated (to check them), executed (e.g. by a process centered environment), and verified (to study their properties)

Using a coordination language for sw process modeling

• Oikos [Montangero et al.] is a multiuser, distributed process-centered software development environment designed and implemented using the ESP coordination language

Oikos is able to execute ESP programs (or similar) that describe a software process; the execution
- reconfigures the system (Oikos is a meta-environment)
- enacts the software process (Oikos is a process-centered environment)
- can be formally analyzed and refined

The working environment consists of a number of tuple spaces that represents working spaces, databases, or coordination channels



• Xerox's CLF has been used to describe software processes [Andreoli Meunier Pagani 1996]

Software architectures

Currently, a main application field for coordination models and languages is specification and design of dynamic software architectures

Definition
A *software architecture* is the structure of the components of a program/system, their interrelationships, and principles and guidelines governing their design and evolution over time.

Building applications using modern distributed platforms implies designing complex software architectures involving very different issues like etherogeneous legacy systems and data structures, security, performance, etc.

We need abstract (coordination) models to help synthesizing these architectures

We need tools to analyze them

We need interoperability and coordination mechanisms corresponding to the models, to simplify the implementation, and supporting the analysis tools, to simplify the verification

Views of architectural styles

• Style as Language

The vocabulary is a set of grammar productions and the configuration rules are the rules of the grammar. Analysis is performed on architectural "programs" in form of checking for satisfaction of rules [Abowd Allen Garlan 93]

• Style as System of Types

The vocabulary is a set of types and analysis is performed exploiting type system as type checking [Aesop]

• Style as Theory

The style is defined as a set of axioms and inference rules, whereas the vocabulary is represented through the logical properties of elements [Moriconi et al 95]

• Style as coordination model

Both vocabulary and the semantic rules are based on a coordination model, for instance the CHAM
[Inverardi and Wolf 95]

CHAM for Software Architecture

The CHAM notation has successfully been used to describe software architecture [Inverardi and Wolf 95]

A CHAM description of a software architecture consists of a syntactic description of the static components of the architecture plus a set of reaction rules to describe the how system evolves in reaction steps

Modularity can be managed with membrane and airlock constructs.

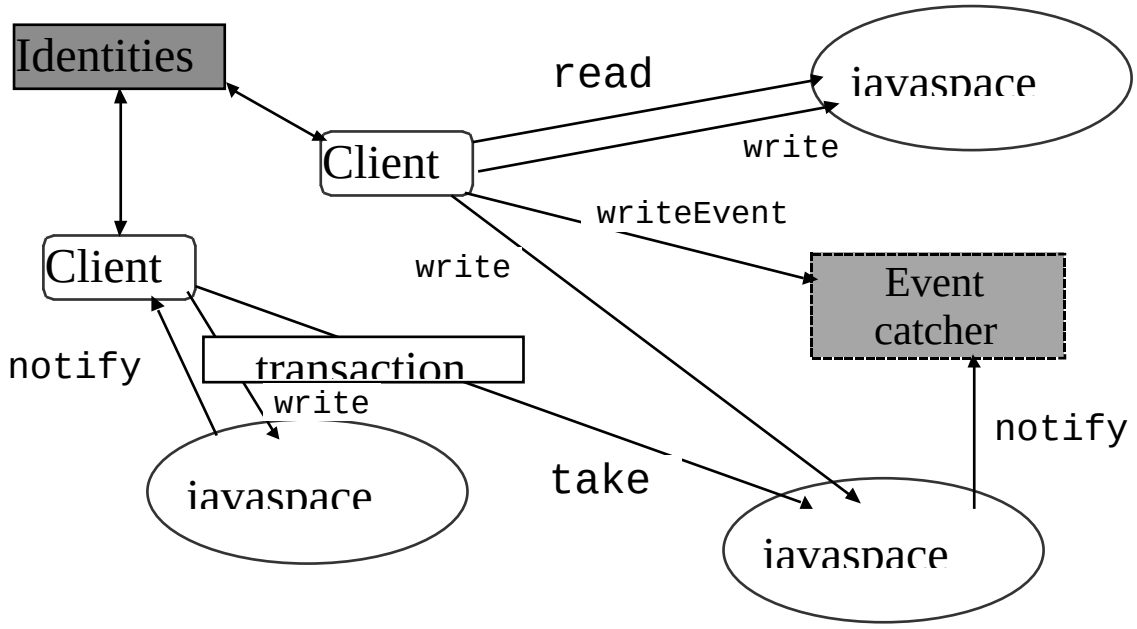
Properties about architecture can be formally proved exploiting mathematical reasoning provided by CHAM.

Some dimensions of coordination

The basic ideas in all coordination models and languages are "*minimalism*"(a small set of coordination primitives should suffice) and "*optimizability*" (it should be possible to reason on and compile coordination primitives)

Coordination models and related languages can be classified along a number of dimensions:

- location-less vs locality-based (named) coordination media
- transactional (multiset based) vs asynchronous (tuple based) coordination media
- procedural (imperative, functional, or logic) vs object-oriented (or agent-oriented) coordinables
- centralized vs decentralized coordination laws
- data-driven vs event-driven coordination primitives



Some current works on coordination

New coordination models and mechanisms
Coordination in scripting languages
Coordination in network-aware languages
Mobile tuple spaces

Implementation of coordination languages
Paradise, LindaSpaces, JavaSpaces
TUCSON, MARS

Specifying and refining coordination programs
PoliS, MobiS
MobADTL

Semantics Expressivity of Coordination languages
Operational
- classic (SOS, CCS, Petri Nets)
- Process algebras
- Timed Linda-like languages

Coordination architectures and applications
- Coordination-based middleware
- Coordination of XML-based workflow systems
- Multiagent system design based on coordination

Conclusions

Coordination languages are a emerging research field

Coordination models are an important tool for a new class of network-aware applications, in which several agents (humans, tools, programs) have to be coordinated

There are several questions that are being addressed:

⇒ which coordination mechanisms are more expressive and useful?

⇒ which semantic models should be used to study such mechanisms?

⇒ which implementation techniques are best?

⇒ which software architectures match some specific coordination requirements?

⇒ which programming logics can be used to reason about coordination programs?

⇒ which new applications can be developed, which exploit the new technology?