

Open Distributed Processing

Peter F. Linington

University of Kent at Canterbury

Why have an architecture?

- Need to structure design and evolution
- Why is there a problem?
 - Systems are organizationally large;
 - complex management structure;
 - evolving within their lifetime.
- Consider an analogy

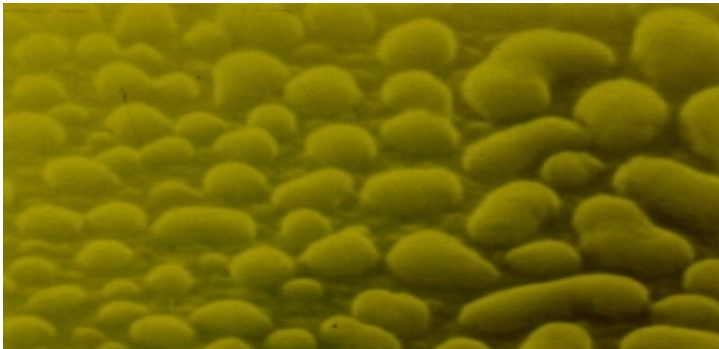


- Silicon on sapphire epitaxy;
- islands of silicon grow by direct deposition and by diffusion;
- islands look like liquid drops.

Stages in Growth



- On a small enough scale, systems that combine have enough time to reconfigure before anything else happens.

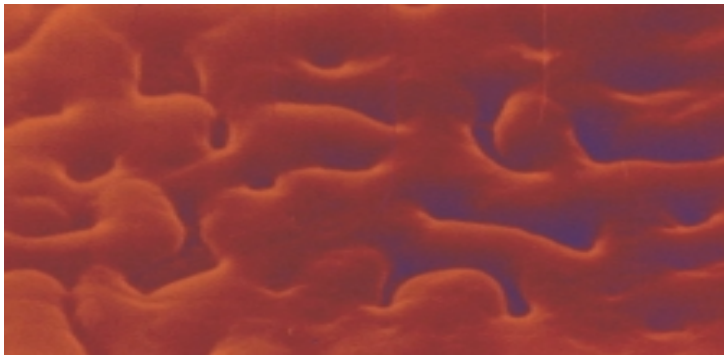


- The bigger the systems are, the longer combination takes, and the more likely it is that reconfiguration will not complete.

Stages in Growth (2)

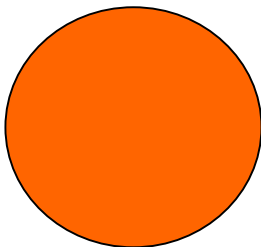
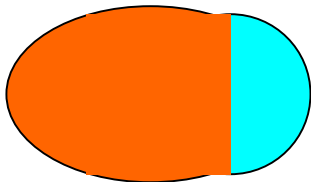
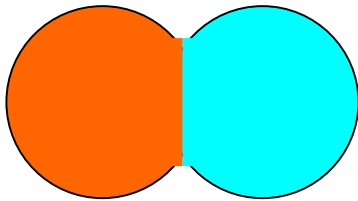
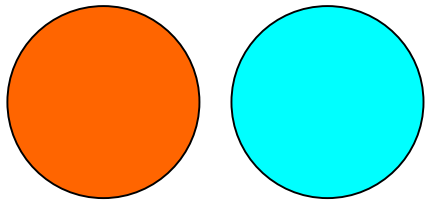


- Once configuration events overlap, long chains of dependencies develop, including transformations.



- Eventually, the chains become linked to form a network;
- finally, residual gaps are filled to form a uniform sheet,

Growth and Reconfiguration



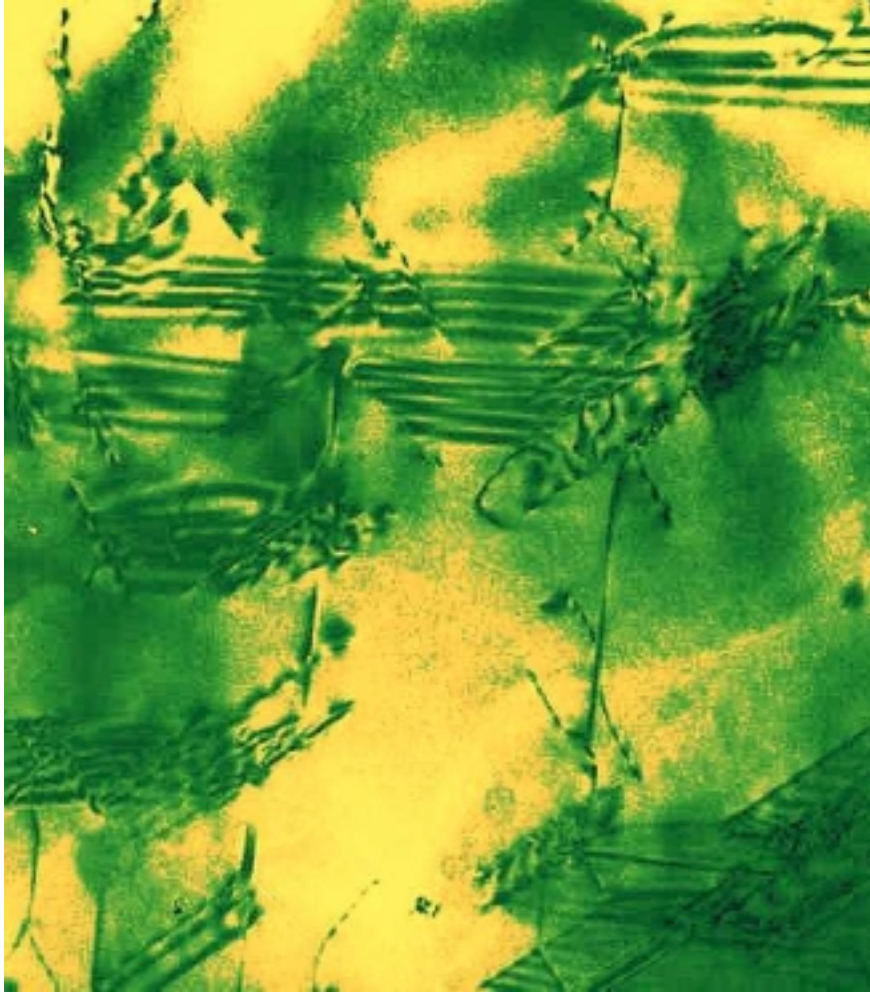
- Independently developed systems have different conventions, and merging them requires
 - a conversion strategy;
 - a migration strategy;
 - eventual elimination of the converters.
- A strong architecture supports and guides such a migration process.

Growth Artefacts

- One consequence of failing to complete each migration process is that it becomes more difficult to phase out transitional structures.
- Complexity leads to frozen-in legacy structure.



Residual Boundaries



- Examining the detailed structure of a large system reveals the boundaries between domains where different conventions were chosen early in the system's life.

Open Distributed Processing

- ODP provides a reference model for system specification and development. It uses:
 - five viewpoints, giving focus for five common areas of concern;
 - a set of engineering functions and optional transparencies as a basis for reuse;
 - a general object-based modelling framework with definitions of basic concepts.

What is ODP?

- Open Distributed Processing (ODP) is a family of standards produced by ISO and ITU-T. It provides a framework for the design of open systems.
 - Reference Model published in 1995 (ISO 10746);
 - functions for Trading, Type Management;
 - frameworks for Naming, Binding, QoS;
 - an Enterprise Language (ISO CD 15414).
- Operates in close liaison with the OMG.

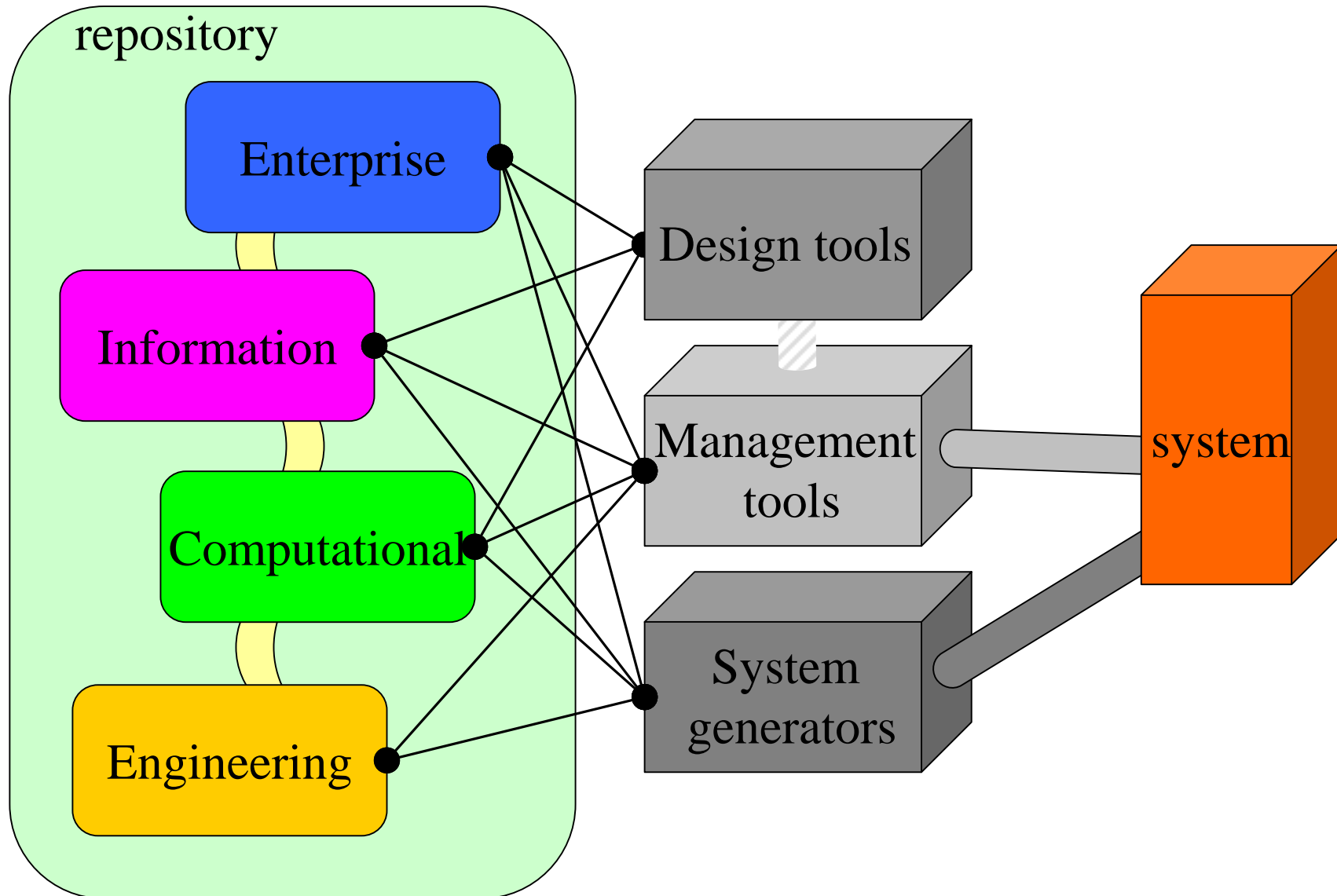
ODP Viewpoints

- The main strategy used in ODP to manage system complexity is introduction of distinct viewpoints.
 - Each viewpoint concentrates on a particular area of concern;
 - Viewpoints are separate but are coupled by correspondences between terms and operators;
 - There are five ODP viewpoints: Enterprise, Information, Computational, Engineering and Technology;
 - Transparencies link Computational/Engineering.

Viewpoints

- The five viewpoints are concerned with:
 - Enterprise - purpose and policy;
 - a strategic view
 - Information - shared data types;
 - a unifying view of information resources
 - Computational - functional application design;
 - an application designer view
 - Engineering - structure of middleware;
 - a view of communication and resource control
 - Technology - known standards.

ODP and the Tool Chain

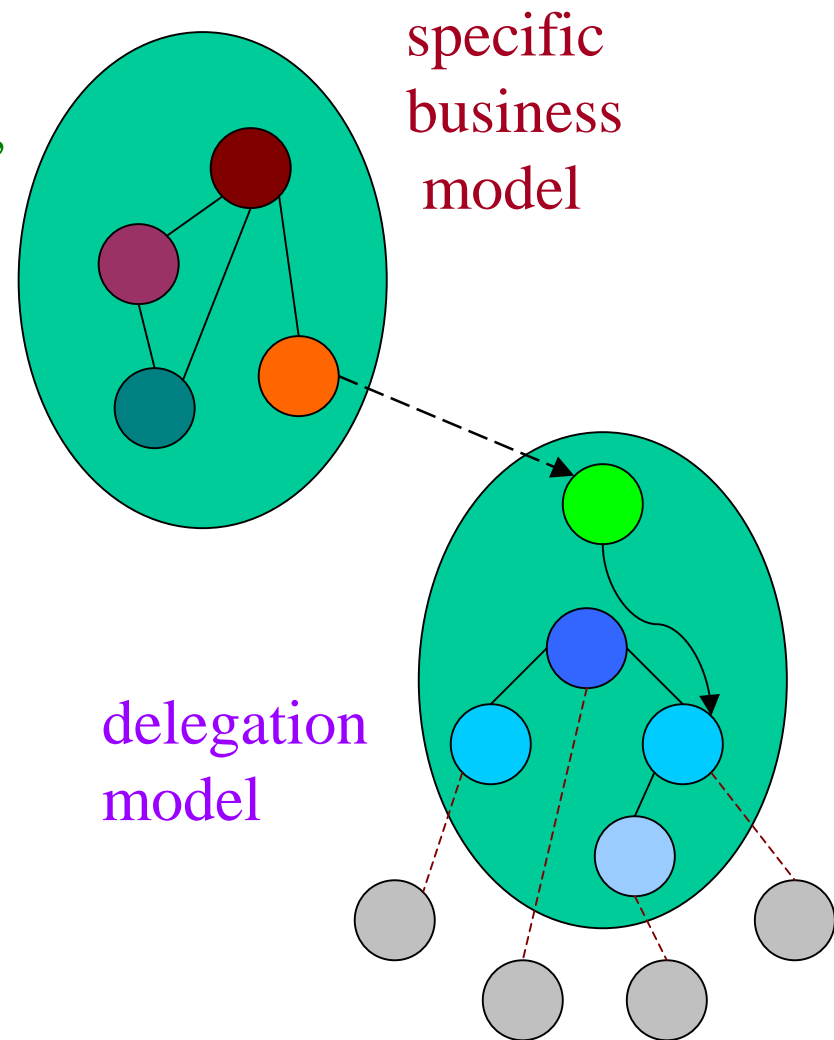


The Enterprise Language

- Defines a high-level view of the environment in which the system is to operate:
 - defines communities of stakeholders;
 - defines constraints on those stakeholders;
 - expresses stakeholders as roles filled by objects;
 - allows generic communities to be specialized by policies;
 - structures specification as nested or overlapping communities.

Community and Structure

- Specific business logic is defined as one community, and delegation rules are defined as another.
- The two communities are linked by the designer establishing correspondences between roles.



Timing and Versions

- Policies change, and changes need to work through to the running system, often on precise time-scales.
 - a pricing policy, or a tax structure may need to be introduced on a specific date;
 - other development and maintenance will need to be carried out in parallel;
 - recovery and back tracking need to be supported;
 - need preparatory changes and dummy running;
 - implies need for flexible build process and version control.

Enforcing Policy

- There are two styles:
 - pessimistic enforcement:
 - involves checking of permissions for every action by some trusted agent which is directly involved in actions;
 - requires active prompting for obligations?
 - used if trust is low and costs of violation high.
 - optimistic enforcement:
 - relies on objects controlling their social behaviour;
 - backed up by auditing and a system of penalties;
 - used if trust is high and cost of violation low.

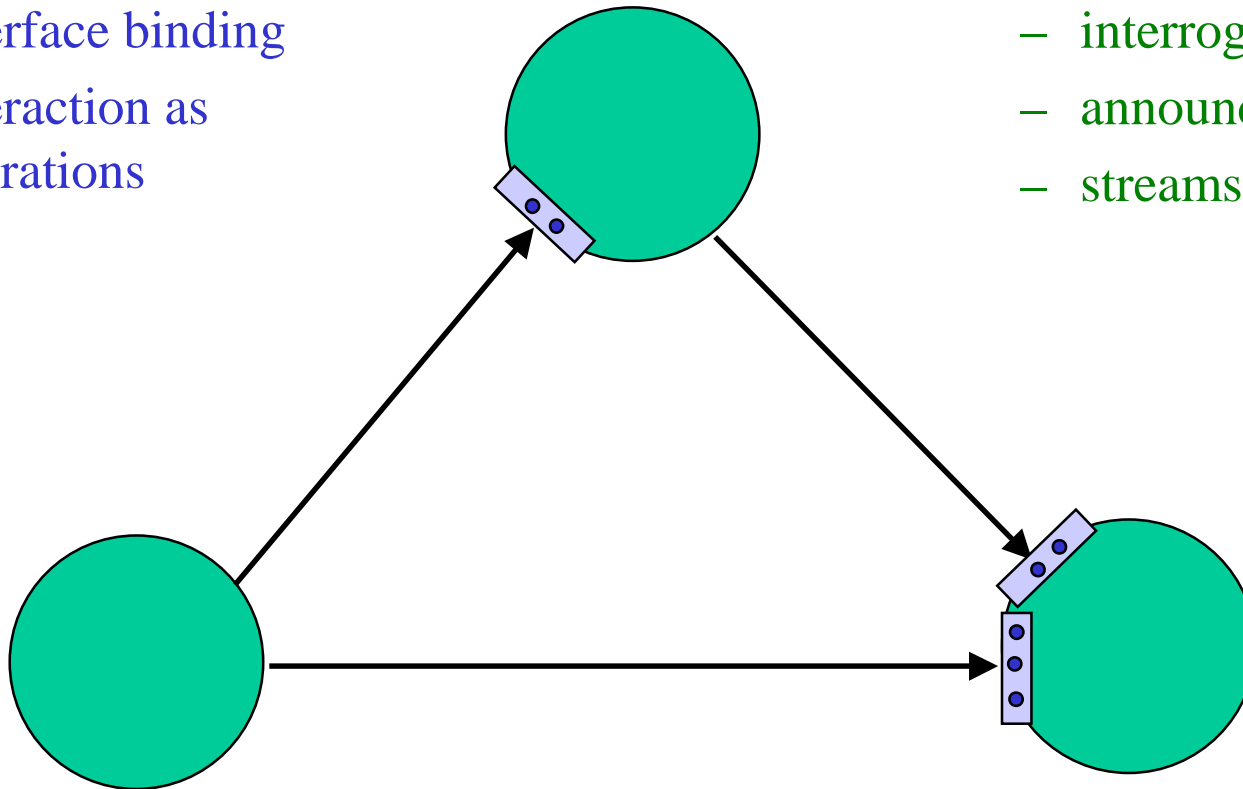
Common Object Model

- ODP is object based;
- the ODP object model starts from observable events and behaviour, with object interaction as the modelling tool;
- objects provide strong encapsulation;
- objects interact at their interfaces; an object can have several interfaces;
- interfaces and objects can be created and deleted (objects delete themselves);
- the ideas of type, template and class support structured specification.

Computational Model

- Object/interface creation/deletion
- interface binding
- interaction as operations

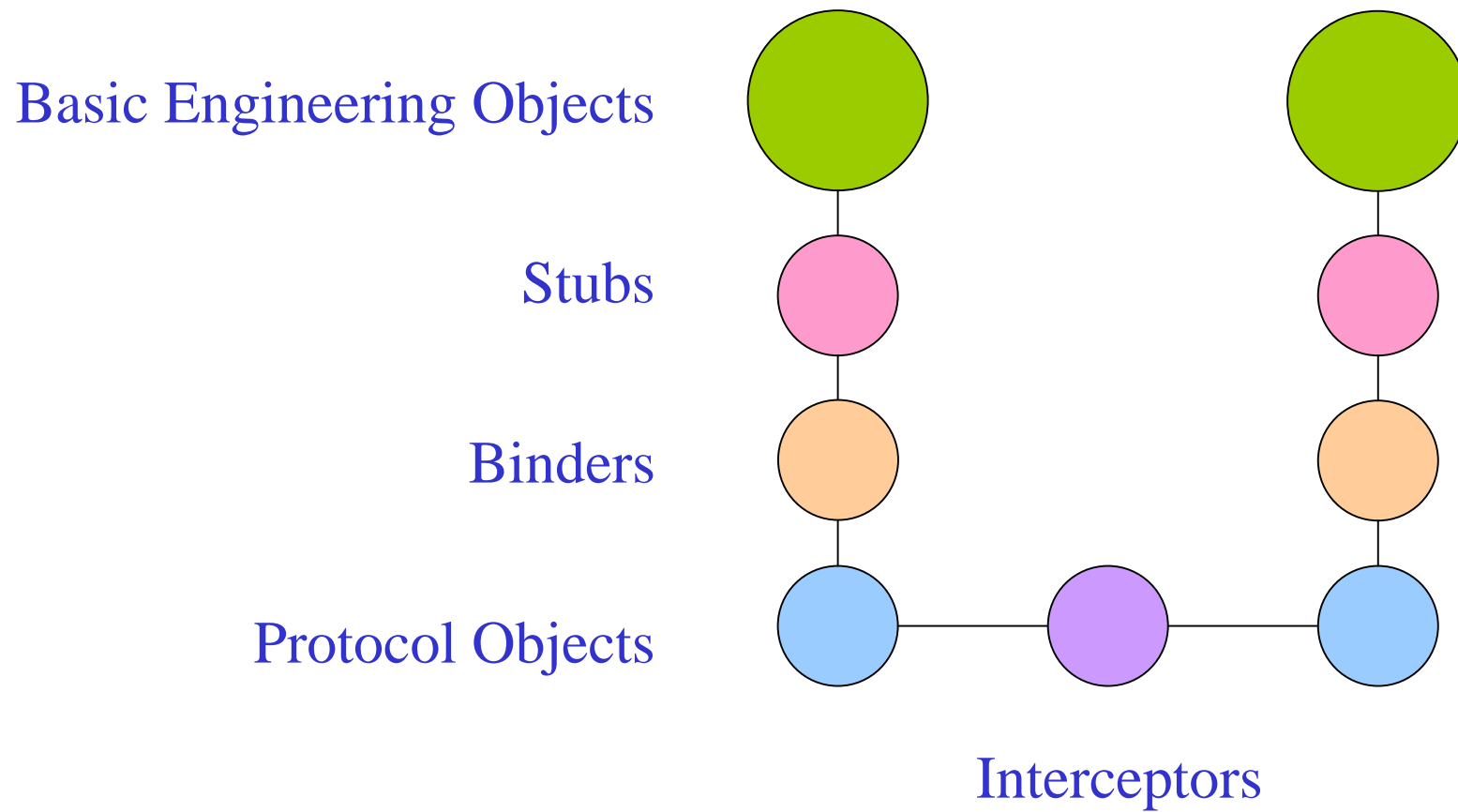
- interactions limited to efficient forms
 - interrogations
 - announcements
 - streams



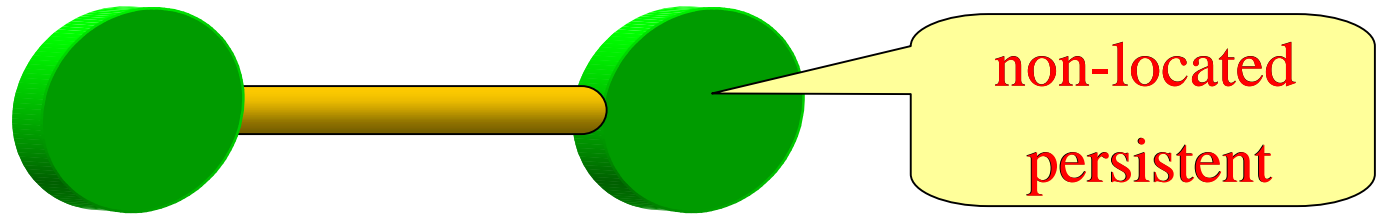
Transparencies

- Hiding, at the designer's request, various adverse effects of distribution.
 - transparencies hide common problems of
 - access
 - relocation
 - failure
 - replication
 - location
 - persistence
 - migration
 - transaction
 - declare requirements for object or interaction properties in a computational or enterprise view
 - there are corresponding standard engineering mechanisms, expressed as templates.

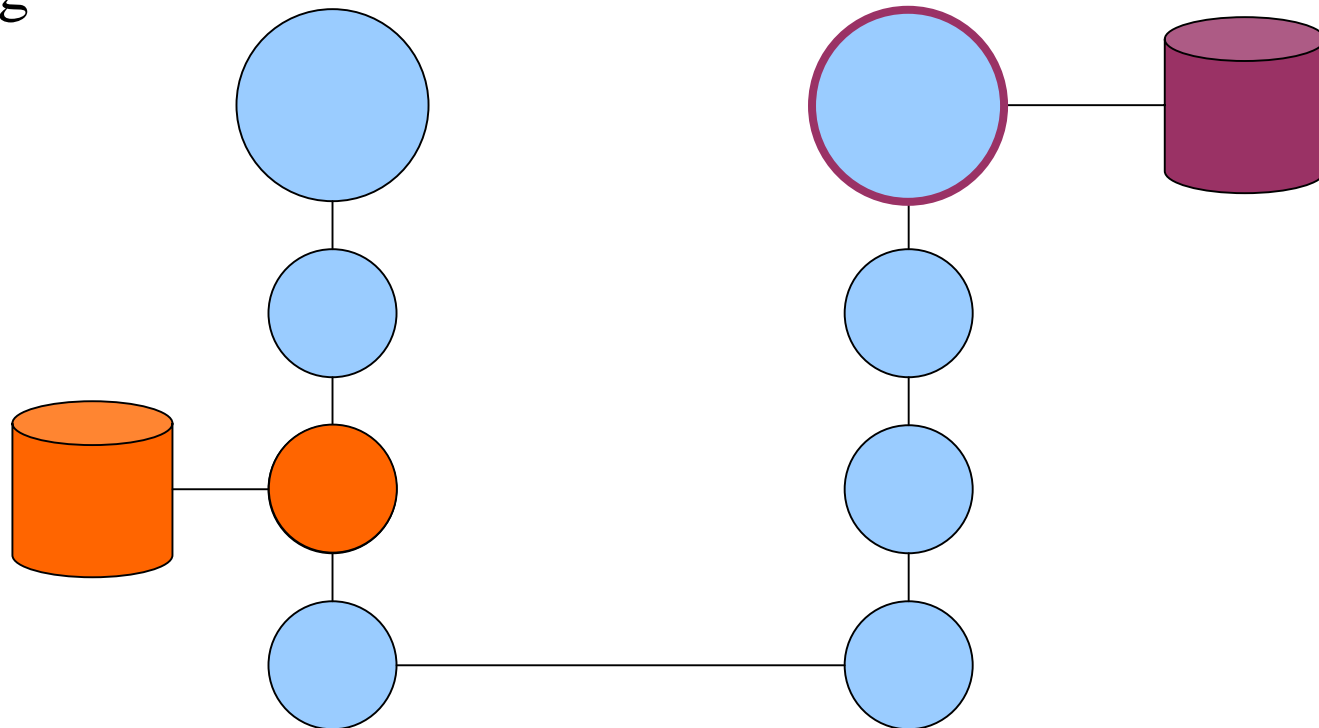
ODP Engineering Channel Model



Applying Transparency



computational
engineering



Enterprise Java Beans

- a practical example of provision of transparency
 - bean packaged with deployment descriptor;
 - includes e.g. persistence type, transaction type;
 - container does the hard work, and has the wider view of how beans interact;
 - bean responds to simple fine-grain calls where bean-specific actions on state are needed.
 - Environment-specific mappings to e.g. back-end databases provided by specific tools.

EJB Transactions

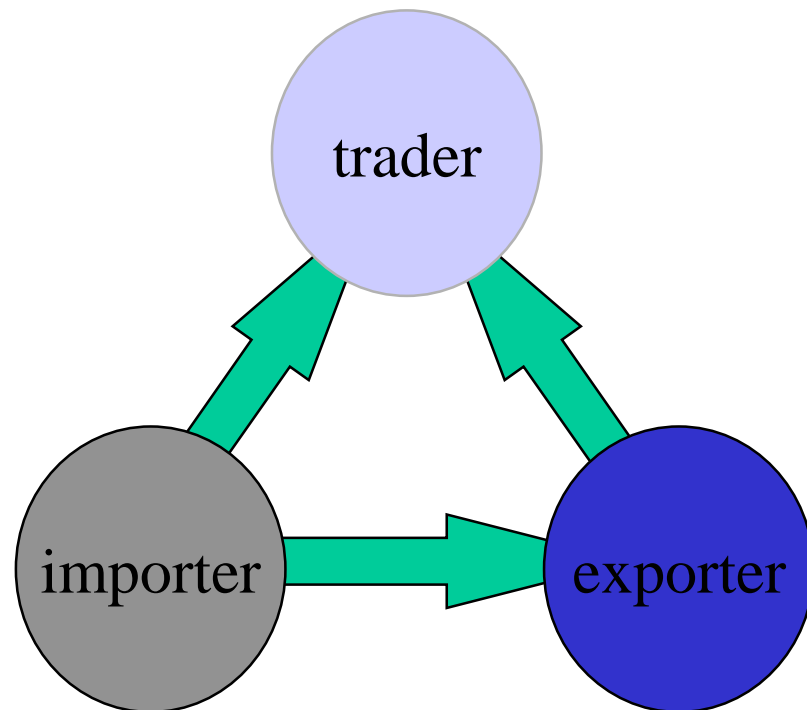
- EJB 1.1, container managed
 - transaction propagation in deployment descriptor
 - Not Supported
 - Supports
 - Required
 - RequiresNew
 - Mandatory
 - Never
 - transaction isolation set by deployer
 - vendor specific details - read uncommitted, read committed, repeatable read, serializable.

ODP Functions

- The ODP Functions are identified in the computational and engineering viewpoints of the RM-ODP. However, when these functions are defined in detail, all the viewpoints can be used. The major groups of functions are:
 - management functions - controlling objects and groupings of objects;
 - coordination functions - providing support and consistency of configurations of objects;
 - repository functions - providing general and specialized storage and retrieval;
 - security functions - building blocks, supporting a range of security policies.

Trading

- offer published by service provider, retrieved by potential clients;
- locate services by type and properties, not name;
- dynamic configuration - finding resources;
- proxy and federation.



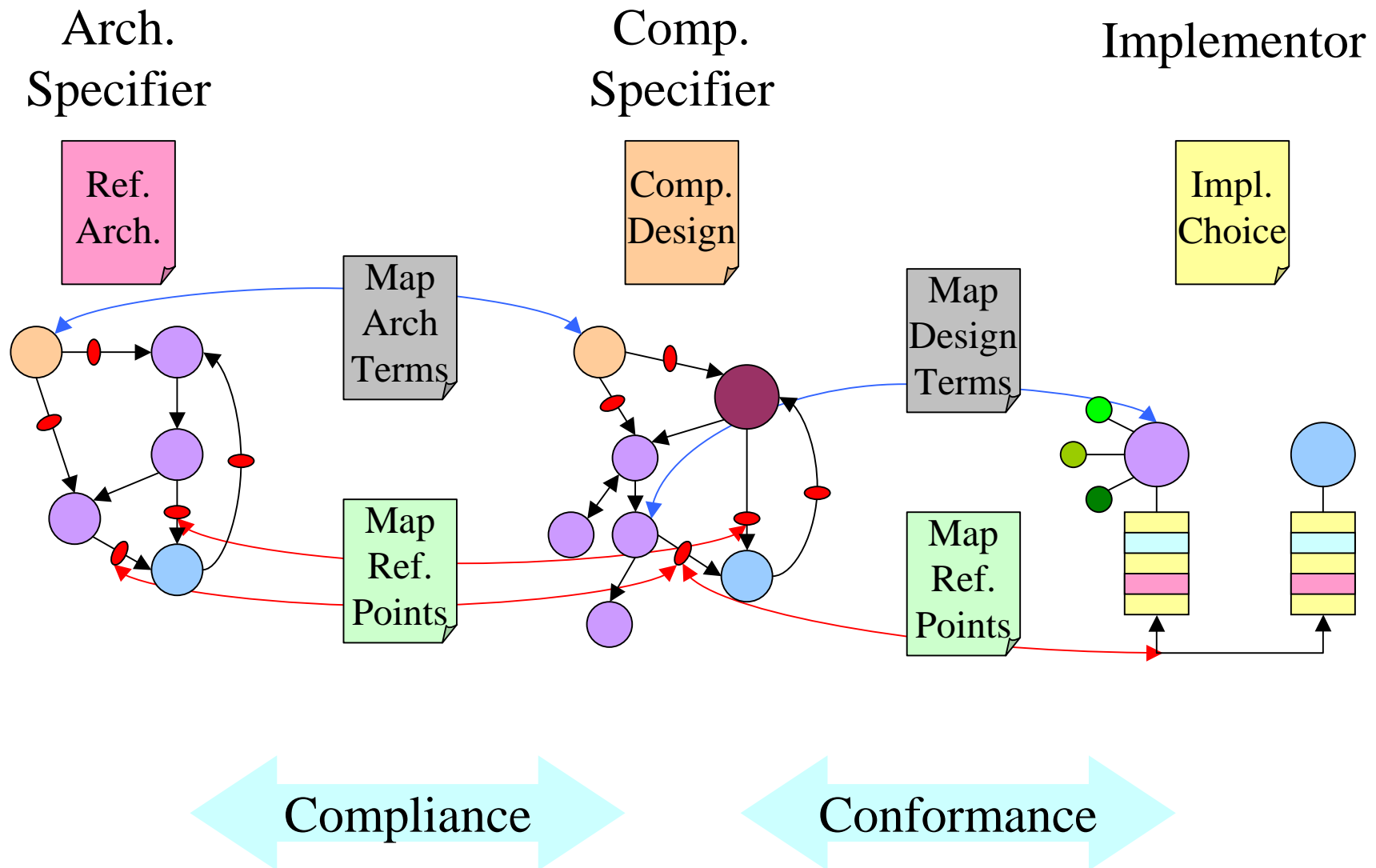
The Type Repository

- Each system has its own set of type definitions, and two systems can only communicate if they share some type information. Types are needed:
 - in trading;
 - to check compatibility during binding;
 - to configure interceptors;
 - to guide compilation and component integration
- The type repository stores
 - details of individual type systems;
 - common features of related type systems.
- Common text with OMG Meta-Object Facility (MOF)

Conformance

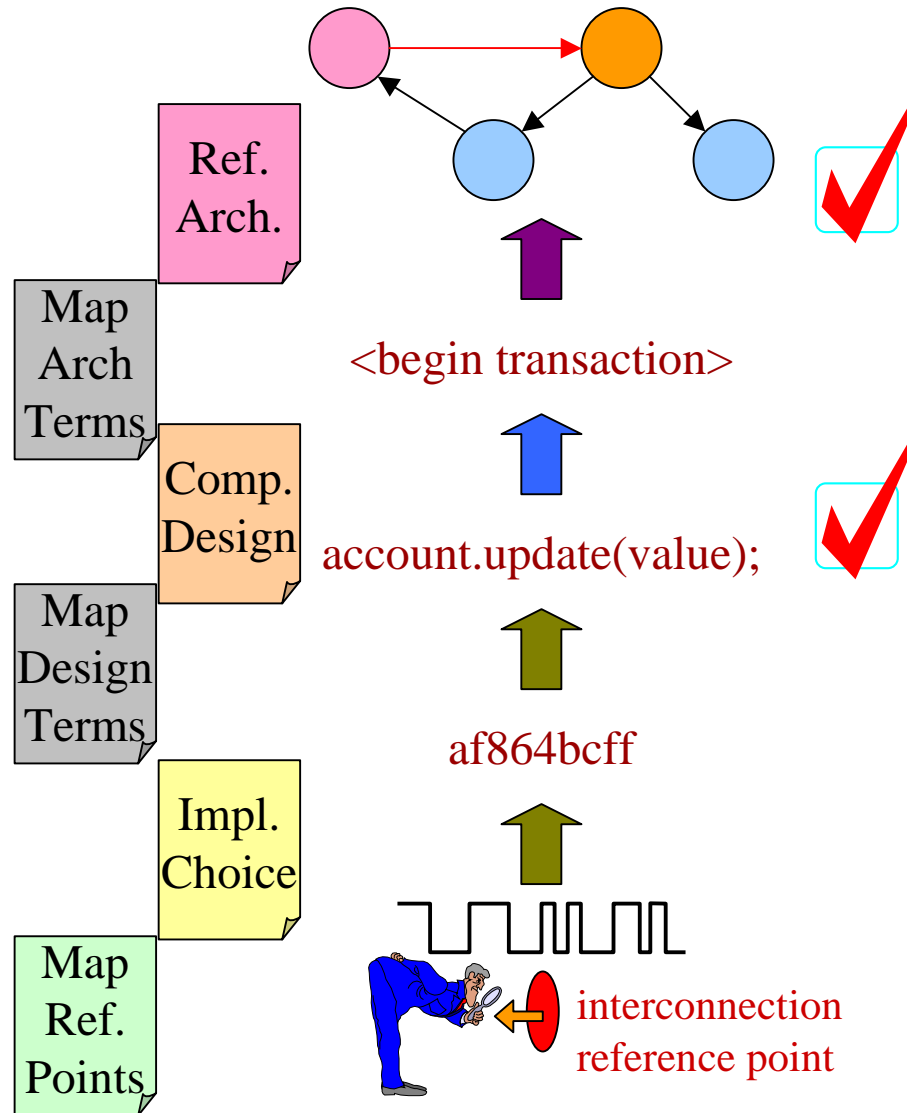
- An architecture without a conformance framework is meaningless; there is no way of saying if it is applied correctly
- Need to involve
 - the architect
 - designers of specific components
 - component implementors
 - testers
 - first, second or third party
 - acceptance, branding, type approval

Conformance & Specification



Conformance and Testing

- Testing is based on observation at ref. points and matching of events to behaviour
- Can observe only at reference points
- Interpret observations using information from implementer until event recognized
- Interpret events in the specification to check behaviour correct.



Object Models and Components

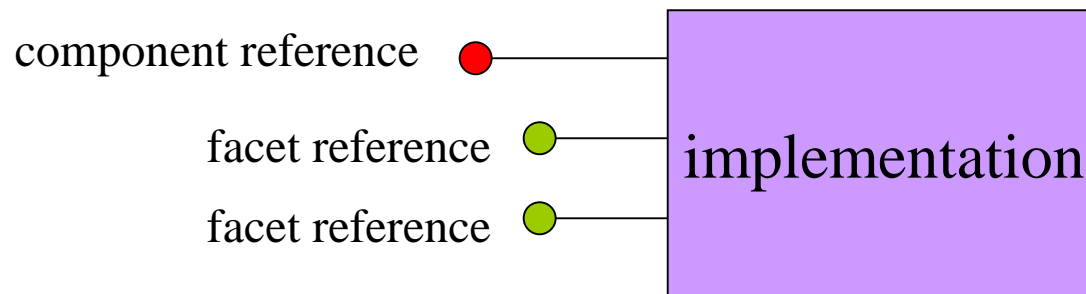
- Issues that lead to different styles of model:
 - What is the objective, communication or portability?
 - Primary function or management?
 - Minimum constraint or guaranteed reuse?
- Typified by question “given a reference to an interface, can you navigate the configuration supporting it?”
- Object models describe the invocation process
- Component models describe the environment in which the component executes.

CORBA Components

- A CORBA Component is an encapsulation of a collection of related pieces of implementation
- Component type both extends and specializes the object type
- Component has a set of surface features (or *ports*):
 - facets: distinct named interfaces provided to clients;
 - receptacles: named connection points, describing the component's use of a reference to some external agent;
 - event sources: named points that emit events;
 - event sinks; named points to which events can be pushed;
 - attributes: named values exposed by accessor/mutator functions - mostly for configuration.

Equivalent Interface

- A Component Reference points to the Component's Equivalent Interface, which gives access to the Component's surface features.
- Given any facet interface, the client can call the *get_component* interface to obtain the component's equivalent interface.
- The client can call the *provide_facet* interface on the component's equivalent interface.
- Via the component's equivalent interface, the client can modify the port states.



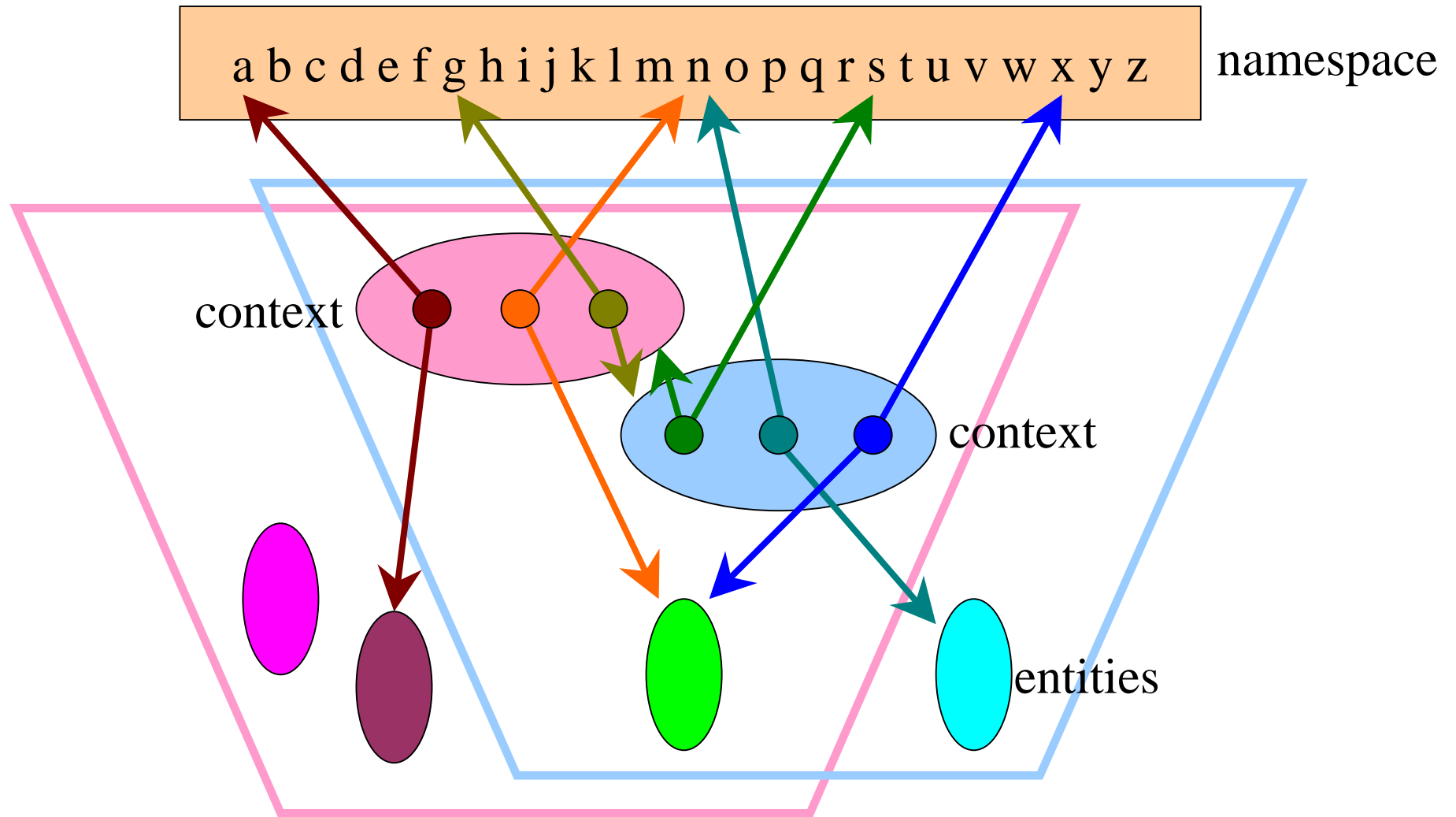
Component Home

- Component Home is a manager that can be used to control a set of components, using the component equivalent interface.
- A Component Home may provide component factory facilities.
- A Component Home can manage a set of specializations of its corresponding component type.
- The component architecture includes a container architecture, defining the environment for supporting dynamic collections of components.

The ODP Naming Framework

- Based on the assumption that all naming is, in some way, context relative
 - defines name, namespace, context, domain, authority, graph, etc.
 - defines name resolution model;
 - gives basis for communication of names;
 - provides ways of organizing naming domains to support the organization of federation.
- So called “global” naming is just a special case.
- Naming is the key to many of the problems of federation and evolution.

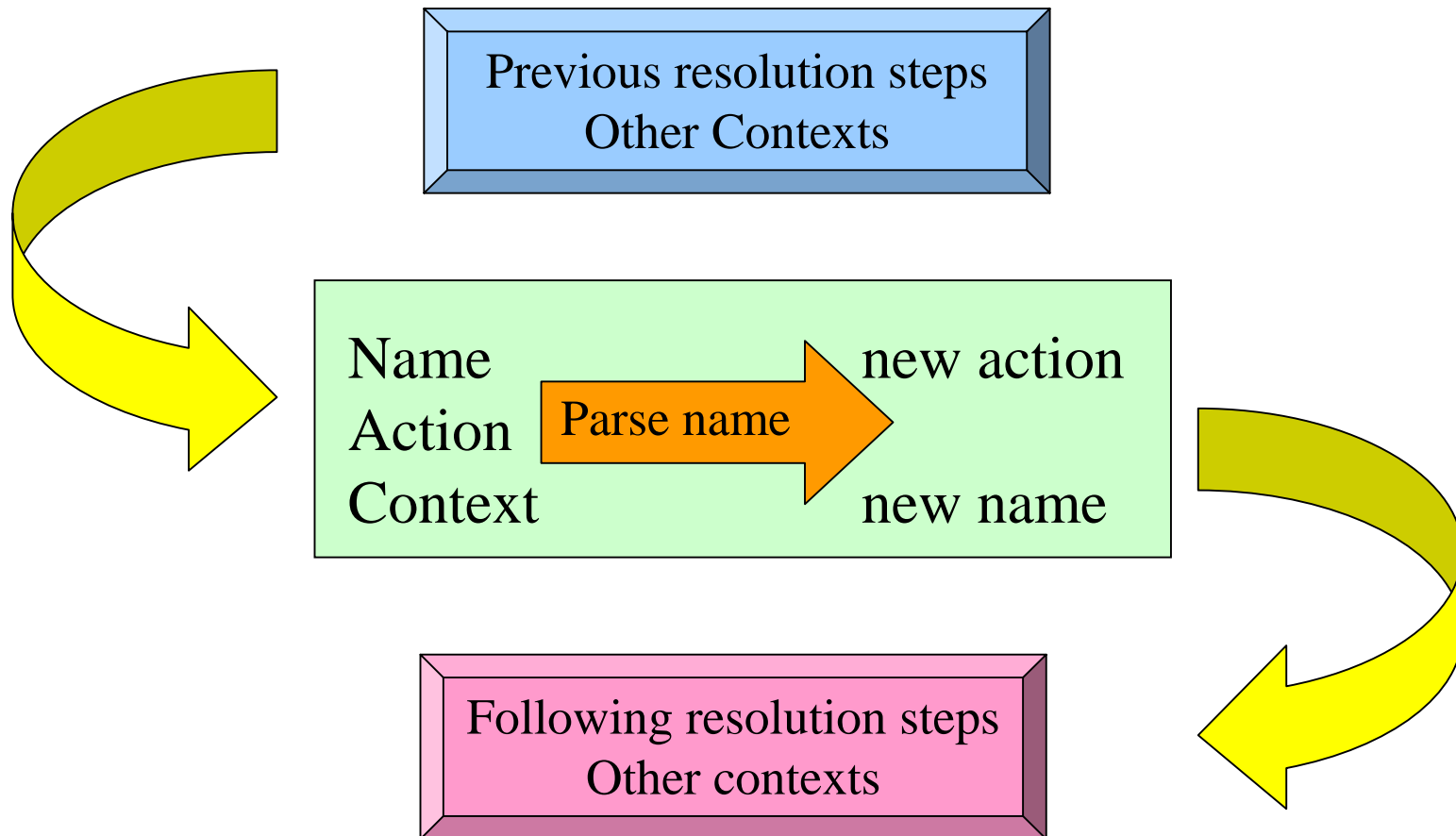
Namespaces and Contexts



Name Resolution

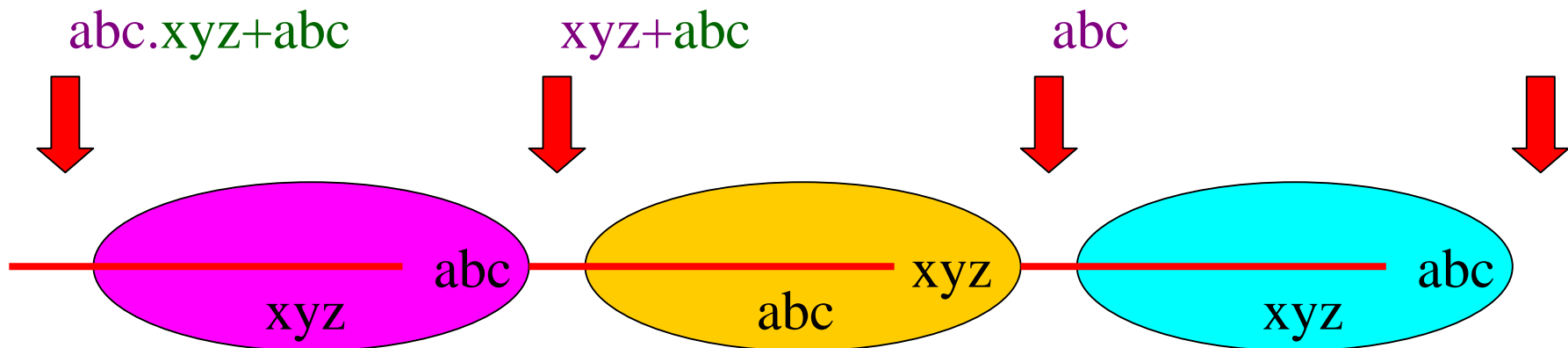
- The process of using a name, involves:
 - the name value;
 - the action being performed;
 - the context in which the resolution is performed
- The result is, in general:
 - a new context;
 - a new value interpretable in that context;
 - a new action to be performed.

Name Resolution Step



Naming Example

- Consider a name associated with a series of actions that creates a path through the network.
 - The name given is `abc.xyz+abc`
 - there are 3 domains; each define `abc` and `xyz`
 - the first delimits with “.”, the second with “+”



Communication of Names

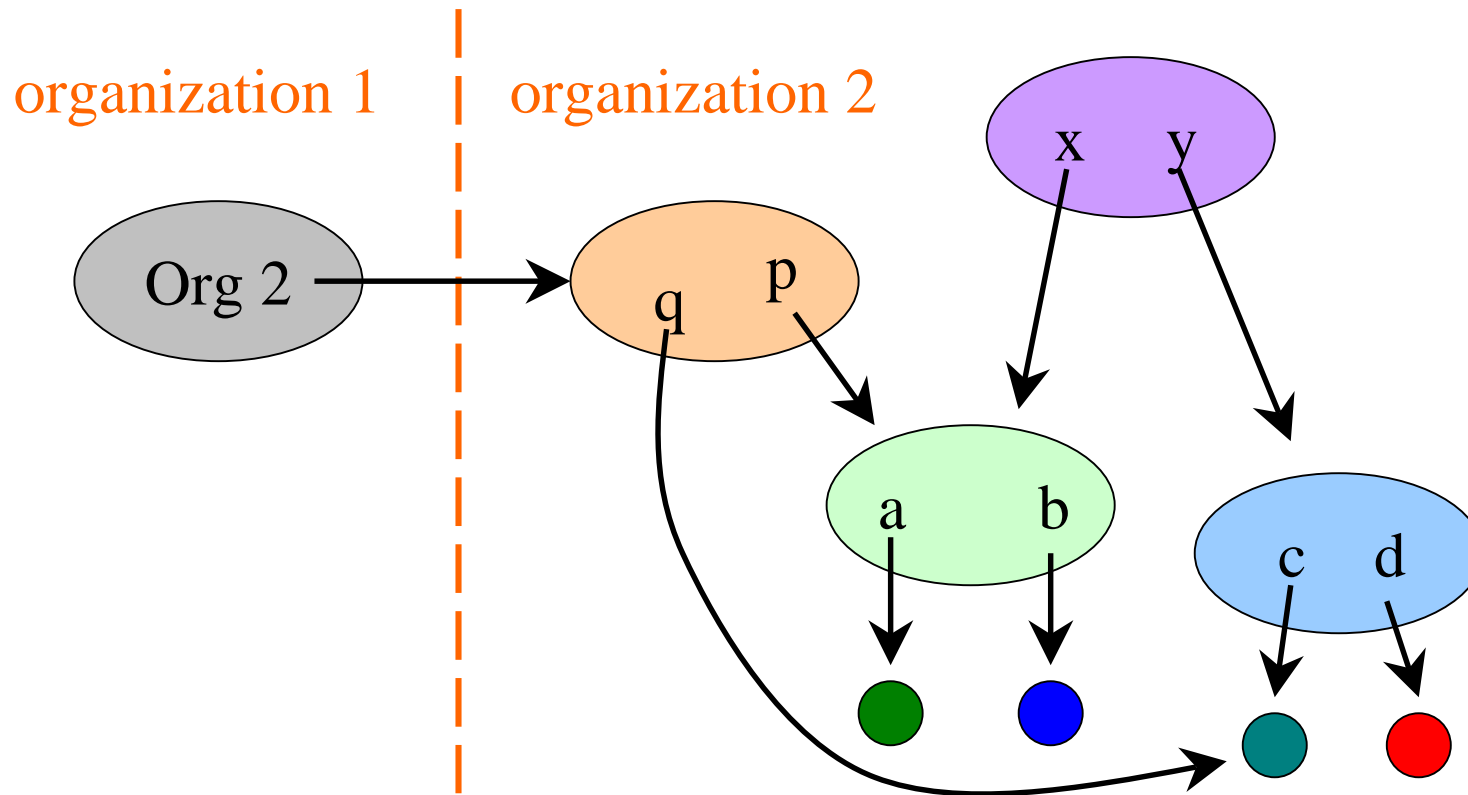
- Communicating a name involves agreement of:
 - a protocol with which to transfer the string of symbols representing the name;
 - context in which to interpret the name.
- The context used can be:
 - shared by parties; may be specific to transfer;
 - primarily of the sender, with the receiver knowing just enough to translate the name;
 - primarily of the receiver, and translated by the sender.

Federation of Naming Domains

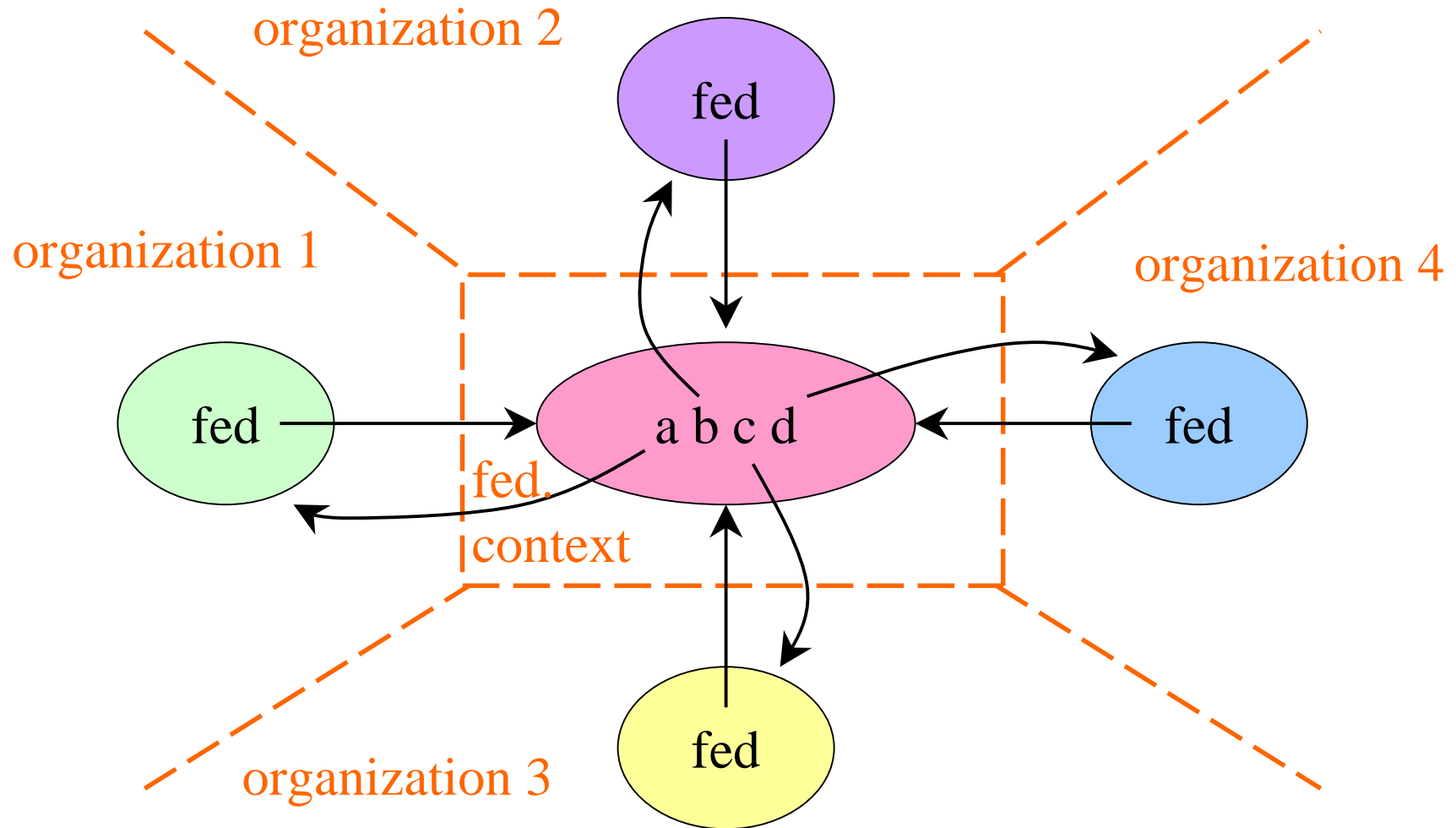
- Support for federation is implicit in context relative naming with step by step resolution, but a number of special styles that simplify federation:
 - an export context, that gathers together published names (and forbids all resolution actions from outside the organization)
 - an import context, that gives local names to hide the detail of federated naming;
 - a federation context, agreed between federation members, that gives uniform shared names.

Export Context

- An organization with an export context restricts resolution for outsiders to that context.



Federation Context



Binding

- The binding of objects can be used at a number of different levels of abstraction
 - in the engineering, describing the establishment of a communication path and environment;
 - computationally, describing establishment of mutual knowledge and ability to interact;
 - in the enterprise, describing contracts and the formation of communities.
- Similar kinds of descriptive tools are needed in each case, although details of, e.g. negotiation, differ.

Interface References & Binding

- The interface references and binding standard refines the outline already given in the reference model:
 - gives the semantics of an interface reference;
 - provides specific representations for interface references in an OMG CORBA domain;
 - gives a detailed binding architecture, including the abstract binding protocol to be used.

Content of Interface Reference

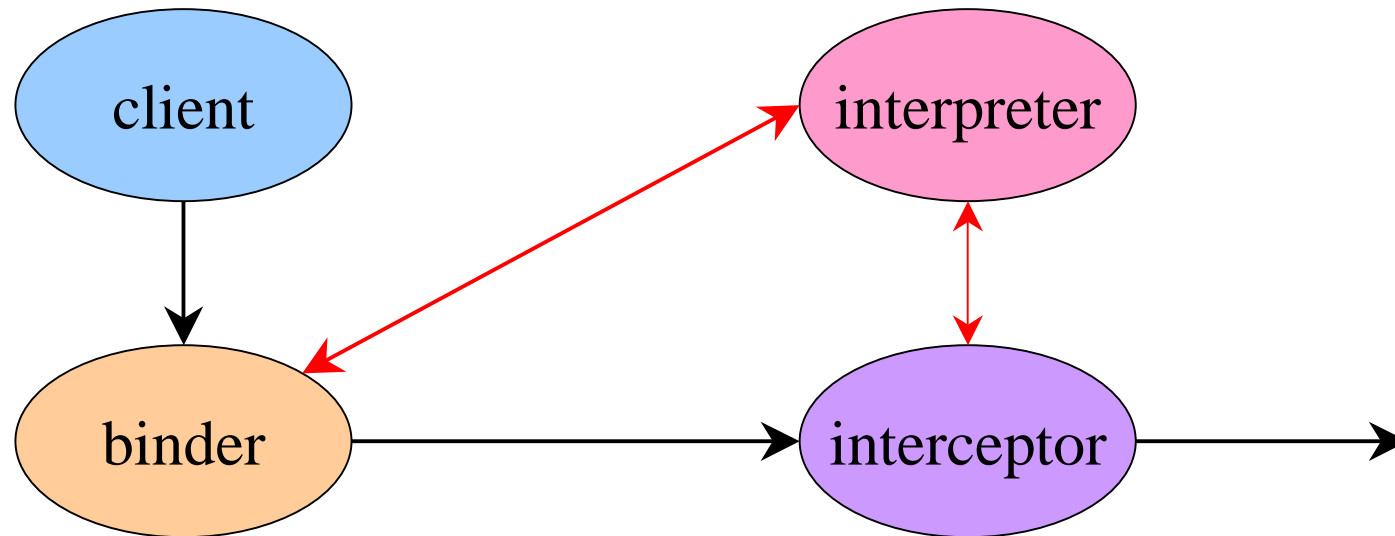
- An interface reference includes:
 - the interface type;
 - the channel class;
 - causality or flow information;
 - location information;
 - relocation information;
 - security information;
 - other quality of service information.
- Information can be explicit or by reference.

Direct or Indirect?

- A direct reference can be interpreted immediately by a binder; an indirect reference contains:
 - the direct reference of an interpreter;
 - a body of opaque information the interpreter can decode, returning a new reference.
- The indirect option allows federation between domains using different direct interface reference representations.
- Actions associated with reference passing are also needed.

Indirect Interpretation

- The indirect reference needs to identify an interceptor, which it may need to create.



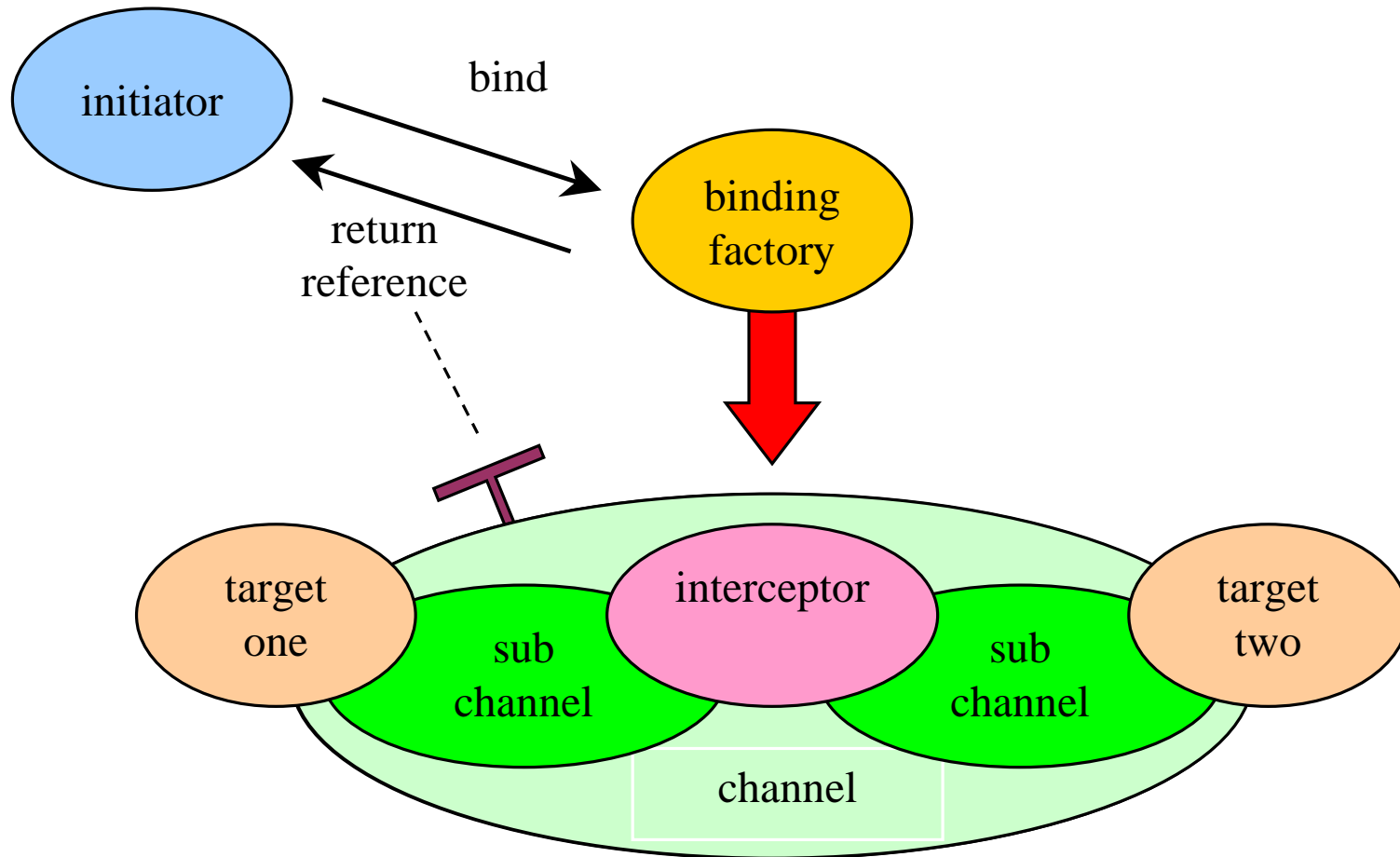
Representing Interface References

- The way that Interface References are represented depends on the middleware in use
 - in CORBA, normal IORs are used;
 - where possible, the defined fields are used, mapping them to an ODP interpretation;
 - additional tag values are introduced to support e.g. indirect references.
- There are potential conformance problems about preservation of tagged values, but these need to be met for other reasons, e.g. IPv6.

Binding Protocol

- The binding protocol involves:
 - location of a binding factory;
 - instantiating by it of a binding channel, including proxies in each node to be involved;
 - creation of supporting objects, such as interceptors, and linking of proxies using them;
 - binding of the target objects to the proxies
 - return of a control interface reference to the creating object by the factory.
- Recursion may be necessary in step three.

Binding Process

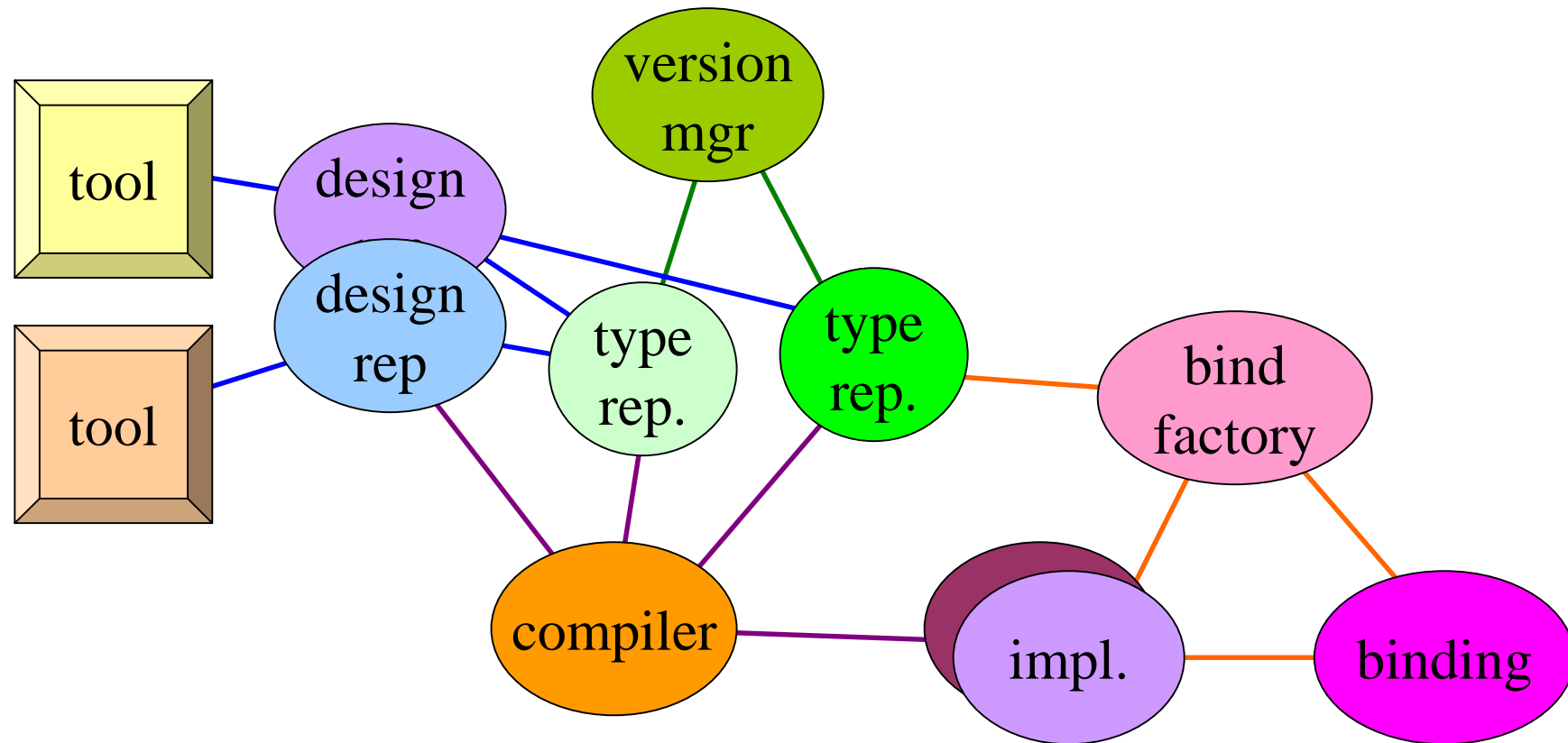


Implications for the Tool Chain

- Change of status of the specifications:
 - they are a resource shared between (providing a path between) many tools;
 - design tools are only one of the things accessing the specifications;
 - the tool chain becomes repository-based;
 - one tool may draw on checking functions of a number of others;
 - there is a need for version control and transaction management.

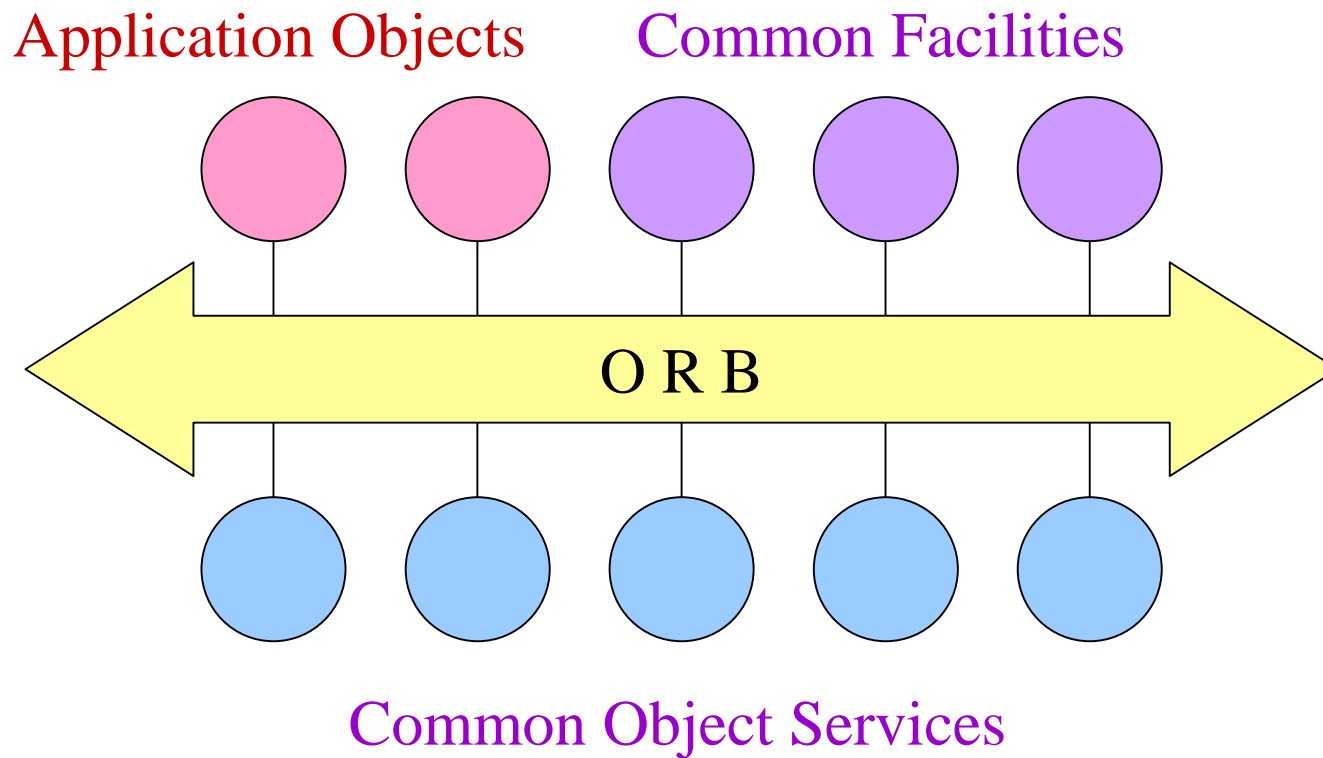
Tool Configuration

- Tools will be linked via different kinds of repository, and can directly affect middleware.



OMG Architecture

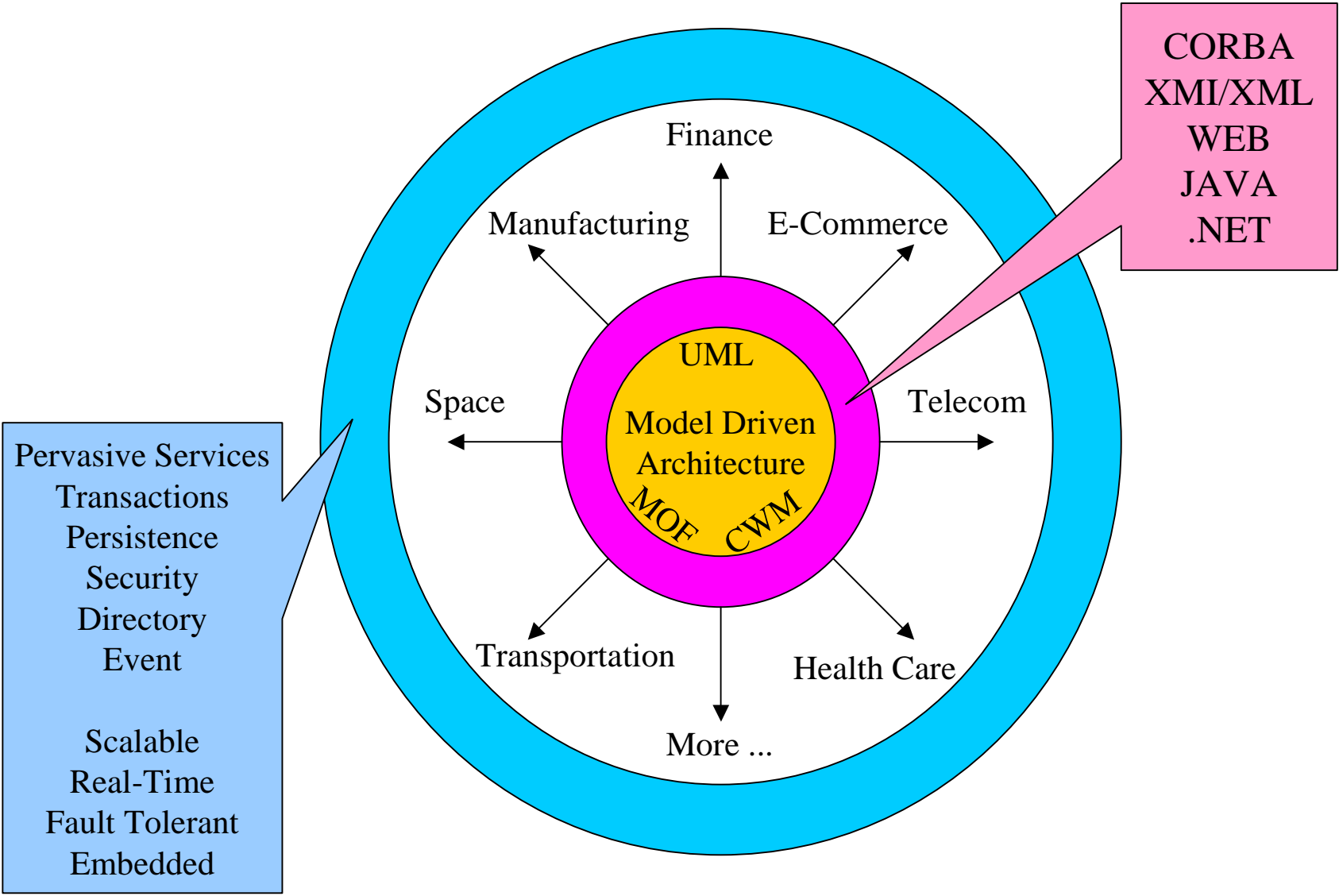
- Traditional OMG Architecture built round CORBA and the ORB.



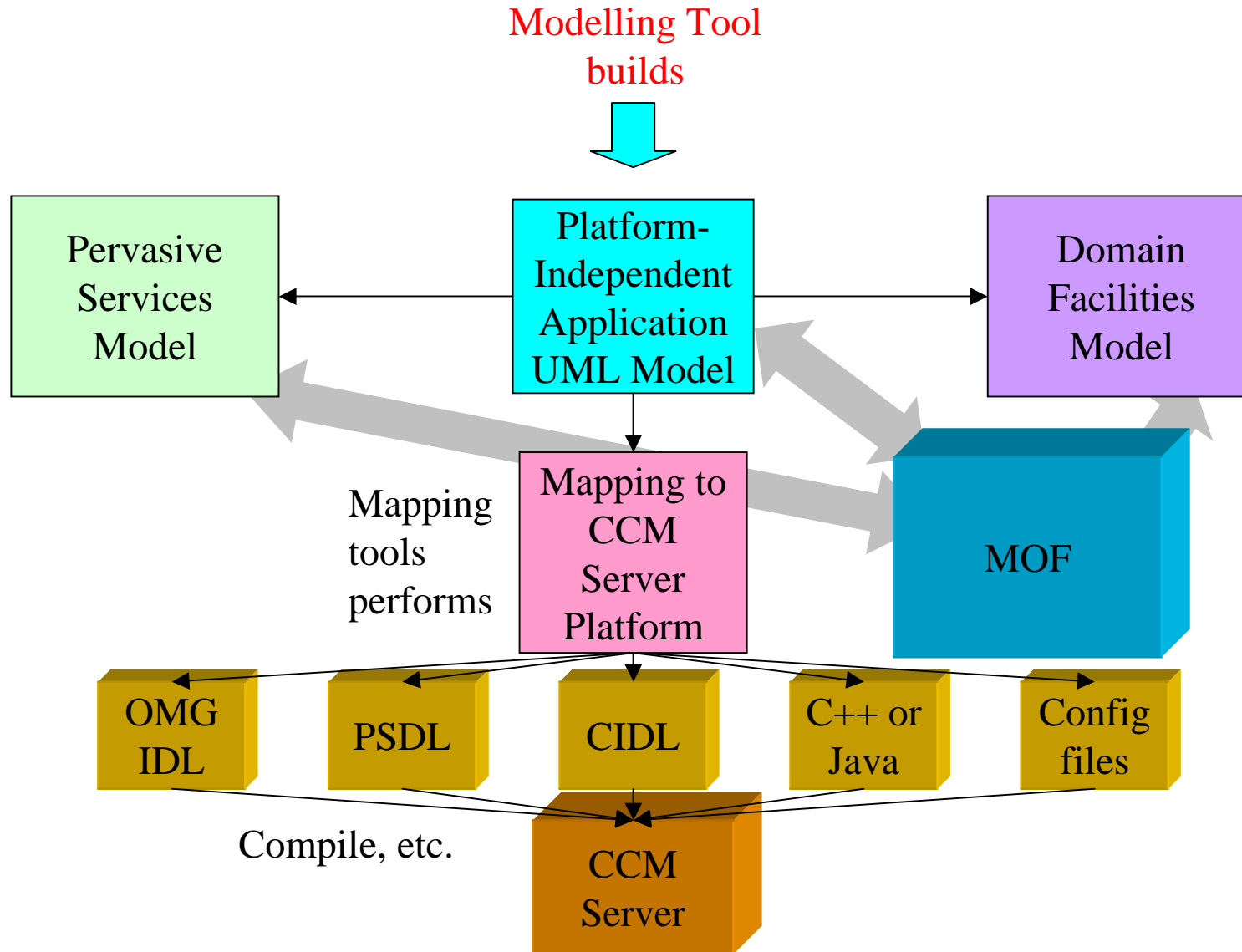
Model-Driven Architecture

- OMG reacting to the facts that
 - CORBA is not the only game in town;
 - JAVA RMI, JINI, Beans
 - DCOM, .NET, SOAP
 - UML is (if they can hang on to change control);
 - the OMA is way out of date and its use is uneven.
- Discovered the need for stable models at a level above middleware.
- Start the tool-chain with set of re-usable UML fragments.

The MDA Positioning Picture



MDA Tool Chain



Conclusions

- What you are building today is a legacy system - you just don't know it yet.
- The more abstract the starting point of your architecture, the more chance of managing evolution.
- Keep formats flexible and expect to carry material you cannot interpret.
- Expect strengthening of the tool chain - capture as much of the requirements material as possible.
- Plan for federation and legacy.