Connecting Plasma to Service-oriented programming with JOLIE

Fabrizio Montesi <fmontesi@italianasoftware.com>
italianaSoftware s.r.l., Italy

Joint work with
Kévin Ottens <ervin@kde.org>
Aaron Seigo <aseigo@kde.org>
KDE team

Thanks to:
Alessandro Diaferia, Sean Wilson [KDE Team]
Claudio Guidi [italianaSoftware s.r.l.]
Marco Montesi [EOS srl]
the Oxygen team [www.oxygen-icons.org]

italianaSoftware

Summary

- What is Service-Oriented Computing (SOC)?
- The benefits that SOC can bring to the desktop.
- What we can do with the current technologies.
- JOLIE and Plasma for a transparent service-oriented desktop.

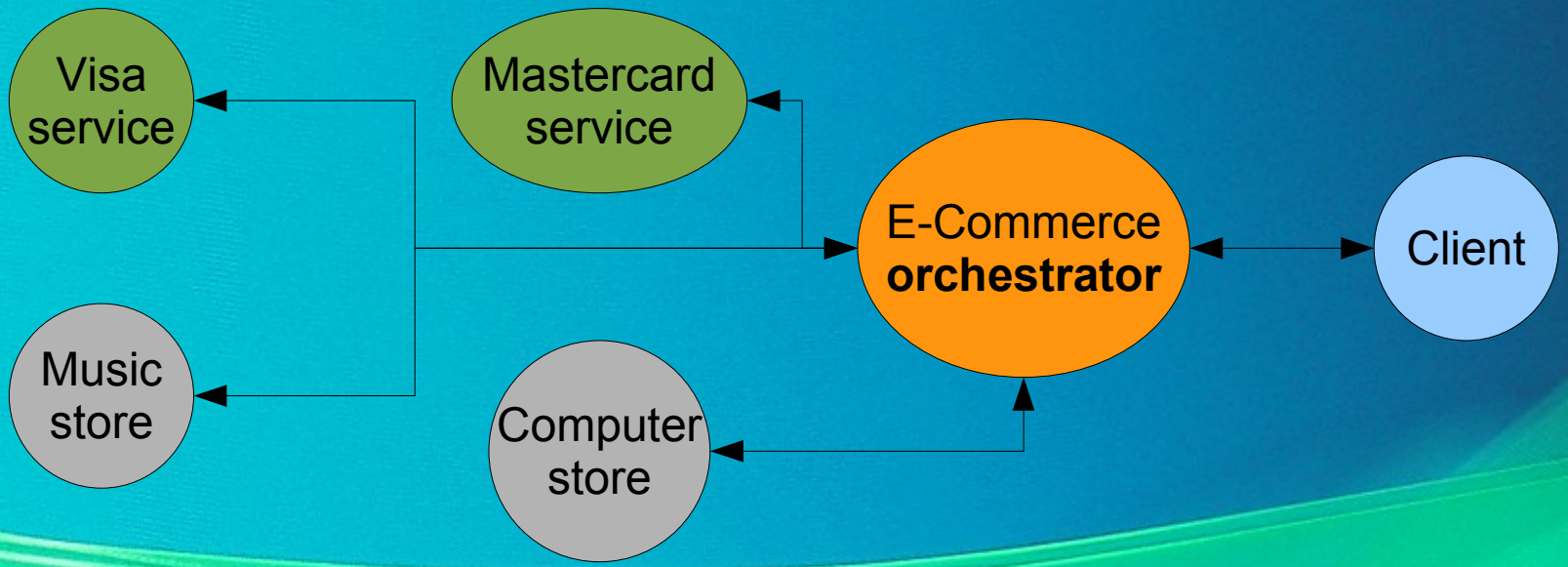
Service-oriented Computing (SOC)

- An emerging paradigm for dealing with distributed applications.
- A new kind of architecture: Service-oriented Architecture (SOA).

- The base component is a service, a software application offering its functionalities by means of an interface.
- Web Services and D-Bus are service-oriented technologies.

Service composition and orchestration

- Service composition allows a programmer to obtain a new service by exploiting existing ones.
- It can be obtained by means of orchestration: an application (the orchestrator) achieves new functionalities by invoking other services.



SOC and the desktop: current state

- Local desktop SOC technologies: D-Bus, DCOP, etc.
- Desktop applications implement ad-hoc mechanisms for interacting with external SOAs.
- Different SOA technologies can not talk to each other:
  - different data protocols: SOAP, D-Bus, DCOP, ...;
  - different transport mechanisms: unix sockets, TCP/IP, ...

Our objective: a service-oriented desktop

- The desktop should be able to:
  - offer its functionalities to other SOAs;
  - exploit the functionalities of other SOAs.
- We want to be able to reuse as many existing services as possible, regardless of their implementing technology.
- Flexible experience: users should be able to compose their desktop with service interfaces as they like/need.

Concept examples: John goes to Akademy

The organizer

John inserts “Mechelen” in a user interface and pushes a button. A service orchestrator composes bank, travel and hotel services to prepare his travel.

The tourist

John takes his internet tablet. A service orchestrator finds out that he's in “Mechelen” and downloads the “Mechelen activity”, a desktop populated with widgets connected to services available in Mechelen. John asks for “beer” and a map pointing to the nearest pub appears.

The presenter

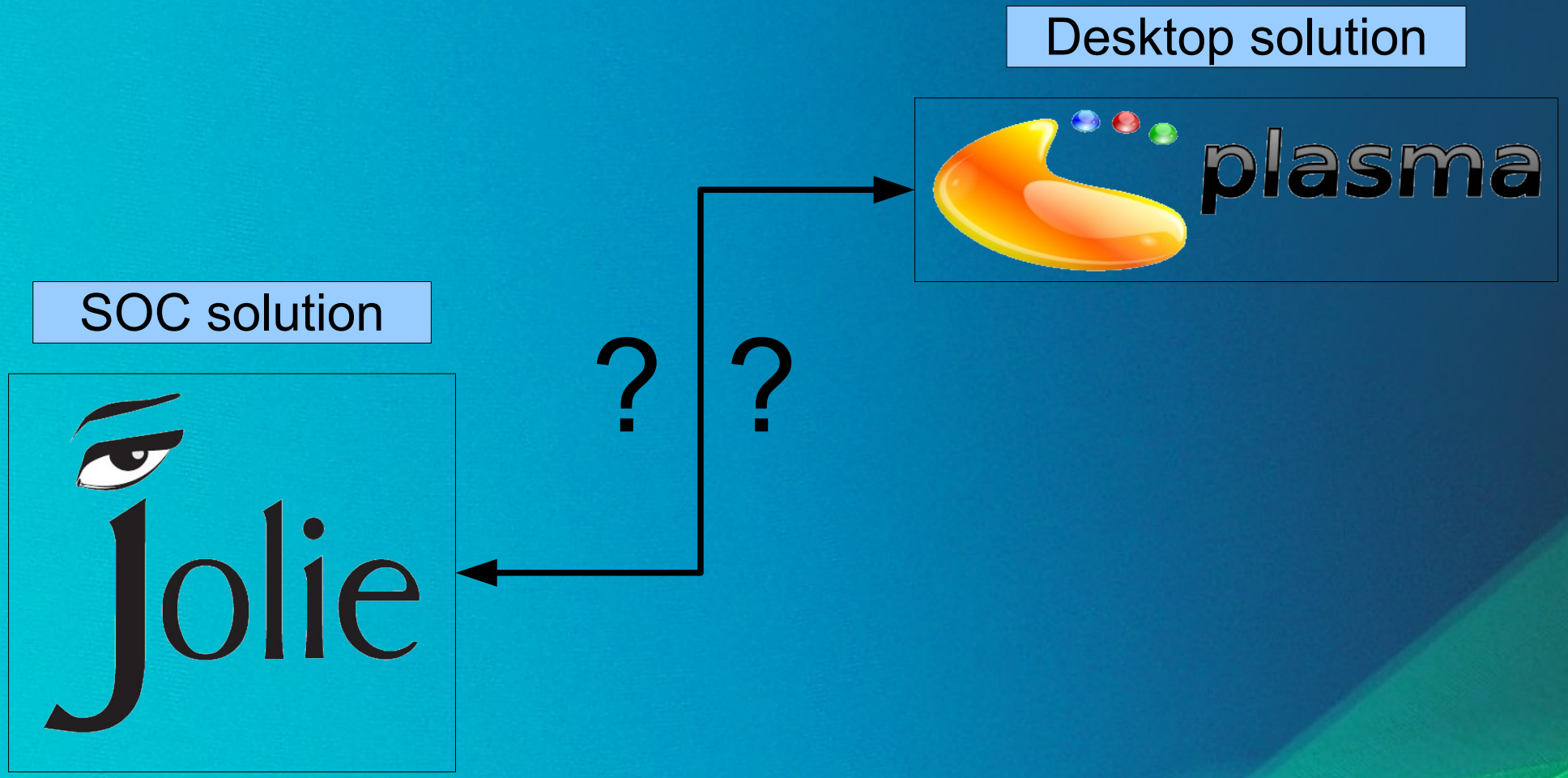
John makes his presentation. Attendees can connect to John's presentation and follow it in their computers.

italianaSoftware

The challenges

- For the SOA inter-connection technology:
  - provide an easy programming environment to create powerful orchestrators;
  - data protocol and transport mechanism independency.
- For the desktop UI framework:
  - separate data from presentation;
  - flexibility: the UI components should be organized by the user.

The right technologies are already here



JOLIE: Java Orchestration Language Interpreter Engine

- Service-oriented programming language.
- Open-source.
- Lightweight and cross-platform.
- Based on a formal calculus for service-oriented computing: SOCK.
- Started as a thesis work at the University of Bologna in the scope of European project SENSORIA.
- Website: http://jolie.sourceforge.net/

A formal theoretical background: SOCK

- A formal calculus for Service-oriented Computing.
- Started as a PhD thesis work at the University of Bologna in the scope of European project SENSORIA.
- JOLIE follows the semantics of SOCK, allowing for formal reasoning on JOLIE programs.
- Every mathematical property proven in SOCK is valid also in JOLIE.
- Example of useful ongoing research: pre-execution deadlock checker for distributed applications.

The challenges

- For the SOA inter-connection technology:
  - provide an easy programming environment to create powerful orchestrators;
  - data protocol and transport mechanism independency.
- For the desktop UI framework:
  - separate data from presentation;
  - flexibility: UI components should be organized by the user.

A full-fledged service-oriented language

- We need to address five issues:
  - communications;
  - data handling;
  - workflow composition;
  - multi-party session handling;
  - fault and compensation handling.
- JOLIE supports all of them.
- We look at a summary of the first three.

Communications

- Communications are native statements, based on operation names.
- Four operation types:
  - one-way: receives a message;
  - request-response: receives a message, executes something and then sends a response back;
  - notification: sends a message;
  - solicit-response: sends a message, then waits for a response.

Communications (2)

```
main {

// One-Way
 setName( name );

// Request-Response
 sum( request )( total ) {
  total = request.operand[0] + request.operand[1]
 };

// Solicit-Response
 getTime@Clock( "UTC+1" )( time );

//Notification
 log@Logger( name + " calculated " + total + " at " + time )

}
```


Data handling

- Easy access and manipulation of structured data.
- Protocols ensure that the structures are translated to the appropriate representation when communicating.

Example: SOAP conversion

JOLIE code

```
person[0].gender = "Male";
person[0].name = "John";
person[1].gender = "Female";
person[1].name = "Ann"
```

SOAP protocol

Converted message

```
<person>
<gender>Male</gender>
<name>John</name>
</person>
<person>
<gender>Female</gender>
<name>Ann</name>
</person>
```

italianaSoftware

Workflow composition

Sequence

```
print@Console( "Hello, " ) ; print@Console( " world!" )
```

Parallelism

```
sendMessage@A( mesg1 ) | sendMessage@B( mesg2 )
```

Non-deterministic choice

```
[ race( name ) ] {
    println@Console( name + " arrived first!" )
}
[ timeout() ] {
    println@Console( "Nobody arrived in time..." )
}
```


The challenges

- For the SOA inter-connection technology:
  - provide an easy programming environment to create powerful orchestrators;
  - data protocol and transport mechanism independency.
- For the desktop UI framework:
  - separate data from presentation;
  - flexibility: UI components should be organized by the user.

Communication abstraction

- JOLIE separates the program logic from the underlying communication details.

```
outputPort Logger {
Location: "socket://www.newlogger.com:810/"
Protocol: sodep
Notification:
    log
}
main {
    log@Logger( "Log message" )
}
```

Dynamic communication configuration

- Service communication configurations can be changed dynamically at runtime.

```
outputPort Logger {
SolicitResponse: log
}
outputPort Registry {
Location: "socket://www.serviceregistry.com"
Protocol: sodep
SolicitResponse: getProtocolForService, getLocationForService
}
```

```
main {
-> getLocationForService@Registry("Logger")( Logger.location );<-
-> getProtocolForService@Registry("Logger")( Logger.protocol );<-
  log@Logger( "Log message" )
}
```

Plasma: the next generation desktop UI

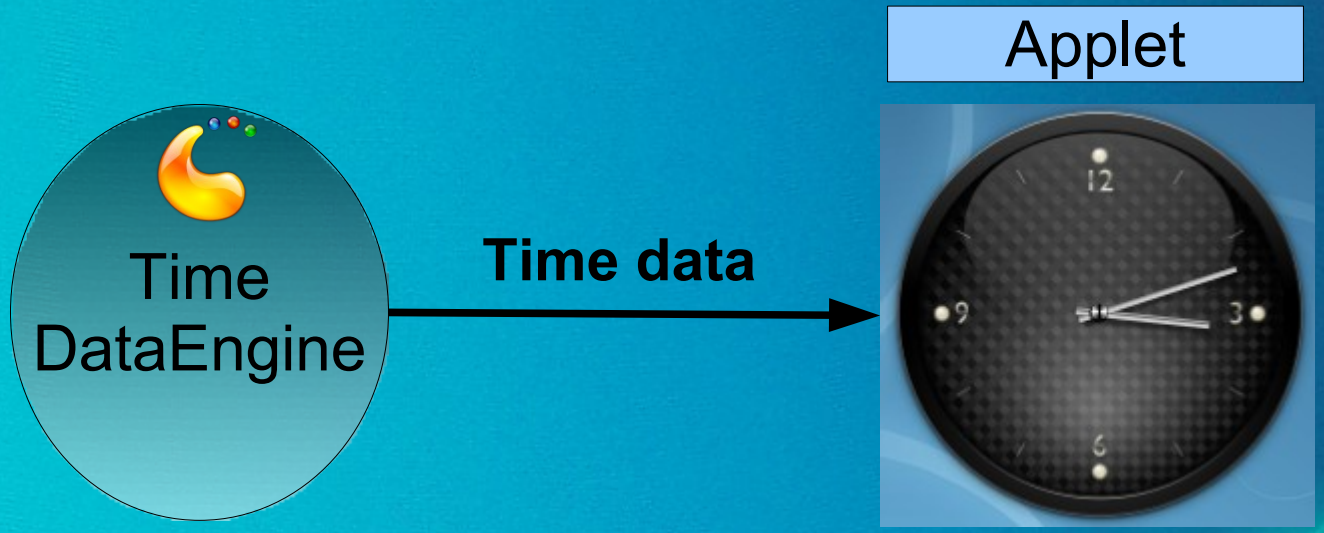
- A flexible User Interface for your desktop.
- Open-source.
- Cross-platform.
- Part of the KDE project.
- Allows users to customize their desktops in new and efficient ways.

The challenges

- For the SOA inter-connection technology:
  - provide an easy programming environment to create powerful orchestrators;
  - data protocol and transport mechanism independency.
- For the desktop UI framework:
  - separate data from presentation;
  - flexibility: UI components should be organized by the user.

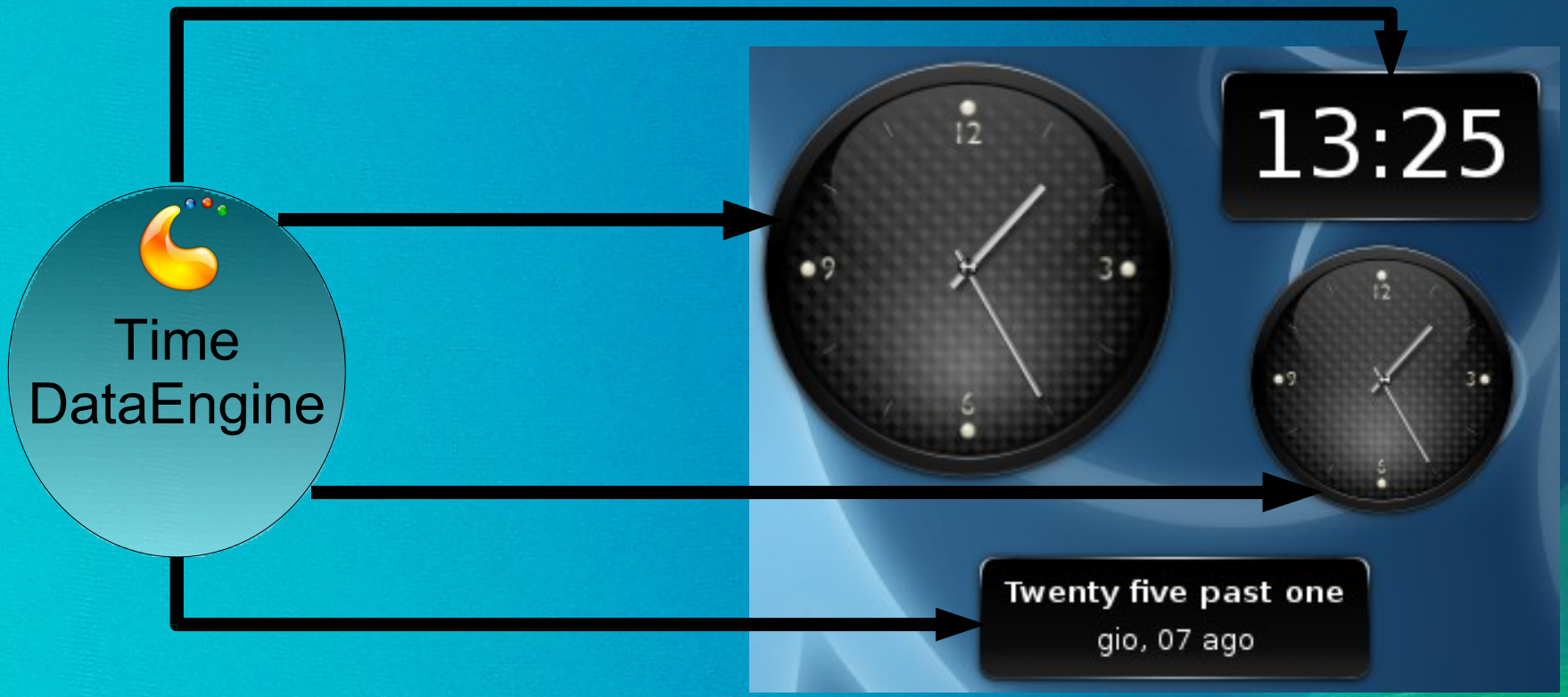
Separating data from presentation

- Plasma provides separated components for data (DataEngine) and presentation (Applet).
- A DataEngine provides Applet access to the data.
- An Applet provides the user a UI to interact with the data.



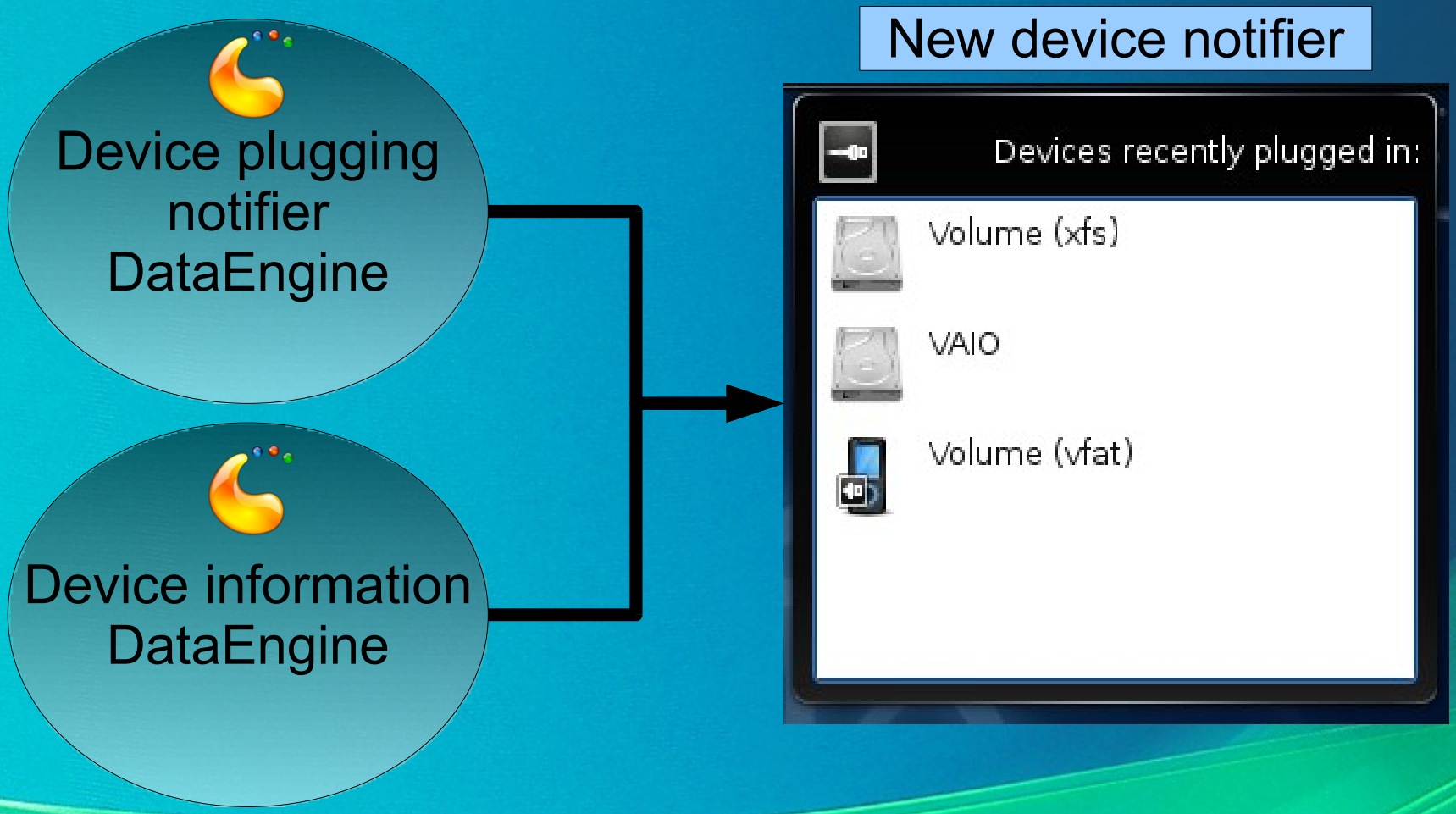
DataEngine sharing

- DataEngines can be used by multiple applets.
- Minimizes resource consumption.



Using multiple DataEngines

- An applet can use multiple DataEngines.



The challenges

- For the SOA inter-connection technology:
  - provide an easy programming environment to create powerful orchestrators;
  - data protocol and transport mechanism independency.
- For the desktop UI framework:
  - separate data from presentation;
  - flexibility: UI components should be organized by the user.

Presentation flexibility

- Plasma allows the user to:
  - place any number of applets wherever he wants;
  - scale applets;
  - rotate applets;
  - group applets in different “activities”;
  - zoom semantically over his desktop, making easy to orientate even in a lot of UI components and component groups.

Placing, scaling and rotating

italianaSoftware

Activities and semantic zooming

The current state

- Two examples of what we can do today.
- We orchestrate D-Bus and DCOP SOAs with JOLIE, forming larger SOAs.

Echoes: managing remote media players

- Echoes offers a web interface for controlling a media player (in this example, we use Amarok).
- The web interface is synchronized with the current state of the media player.
- The state is shared among the clients.

JOLIE web server technology testing in collaboration with:

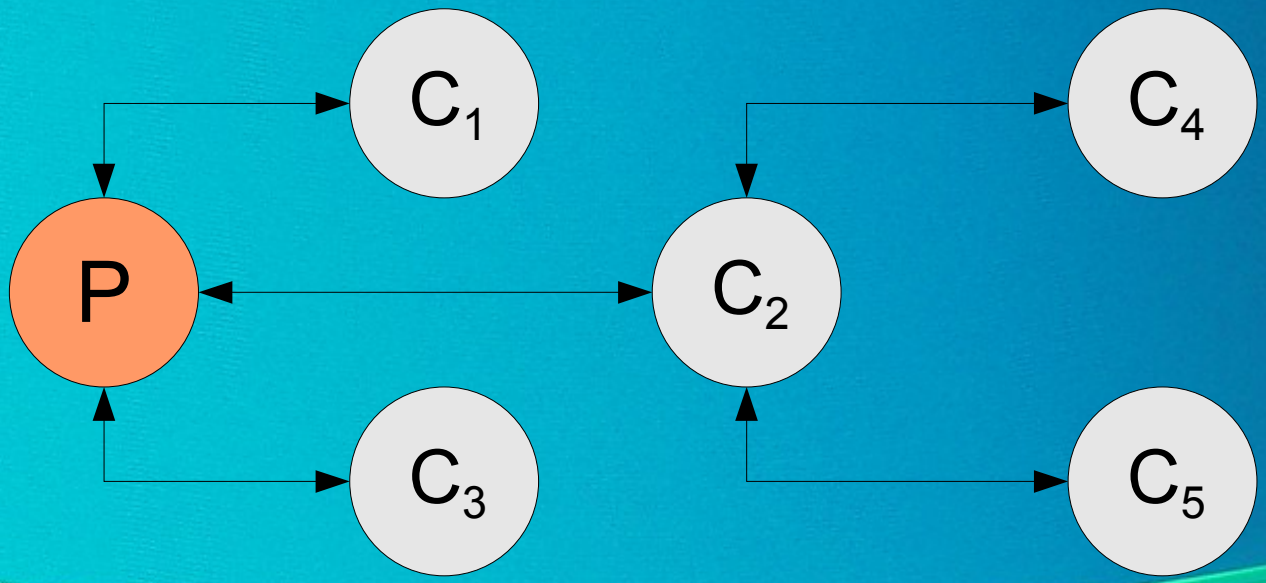
italianaSoftware

Echoes

DEMO

Vision: a SOA for distributed presentations

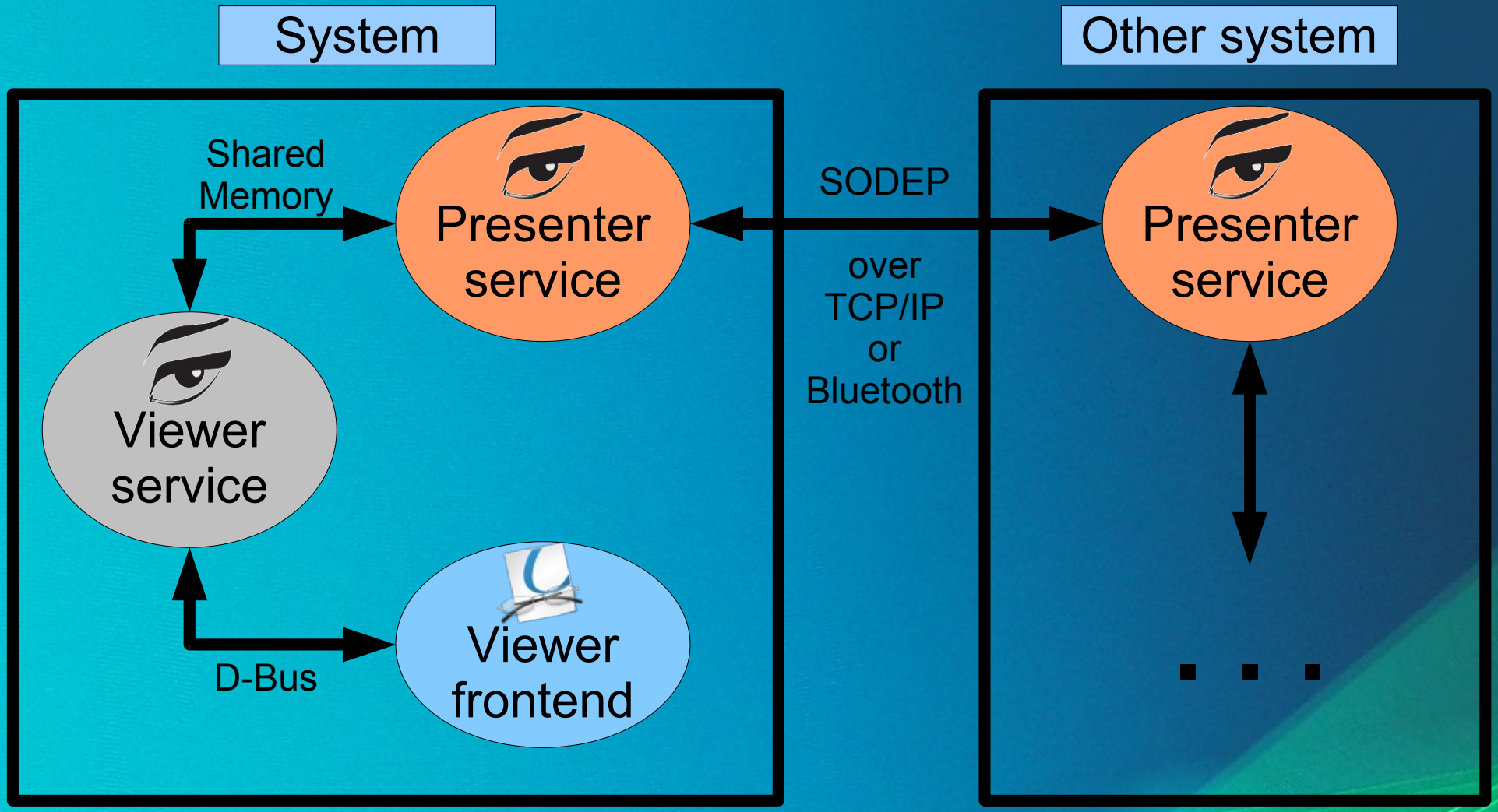
- A presenter (P), gives a presentation.
- Some clients (C_1,C_2,C_3,...) want to follow that presentation in their local viewer.
- The resulting network is a P2P one.



Vision

DEMO

Vision: the architecture

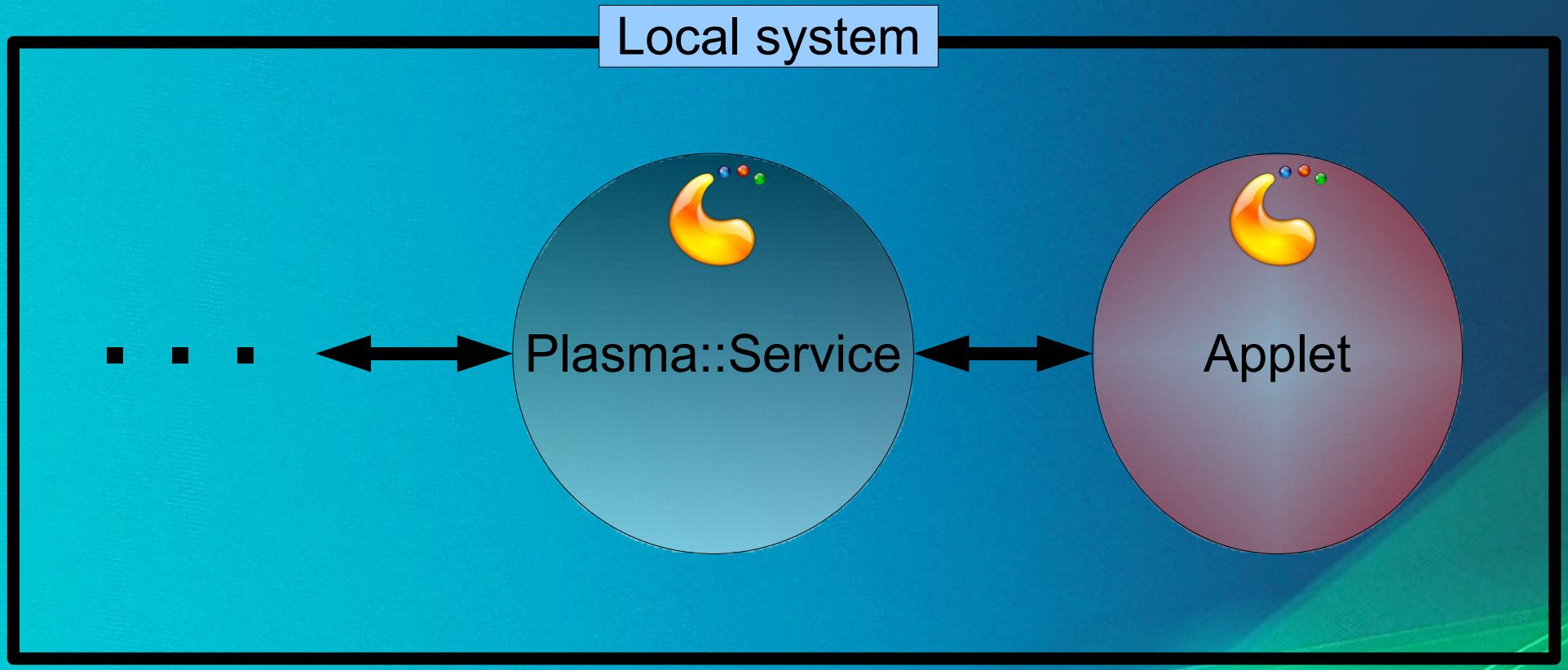


Connecting the two solutions

- JOLIE solves the SOC related problems.
- Plasma solves the UI related problems.
- Let's make them talk to each other!

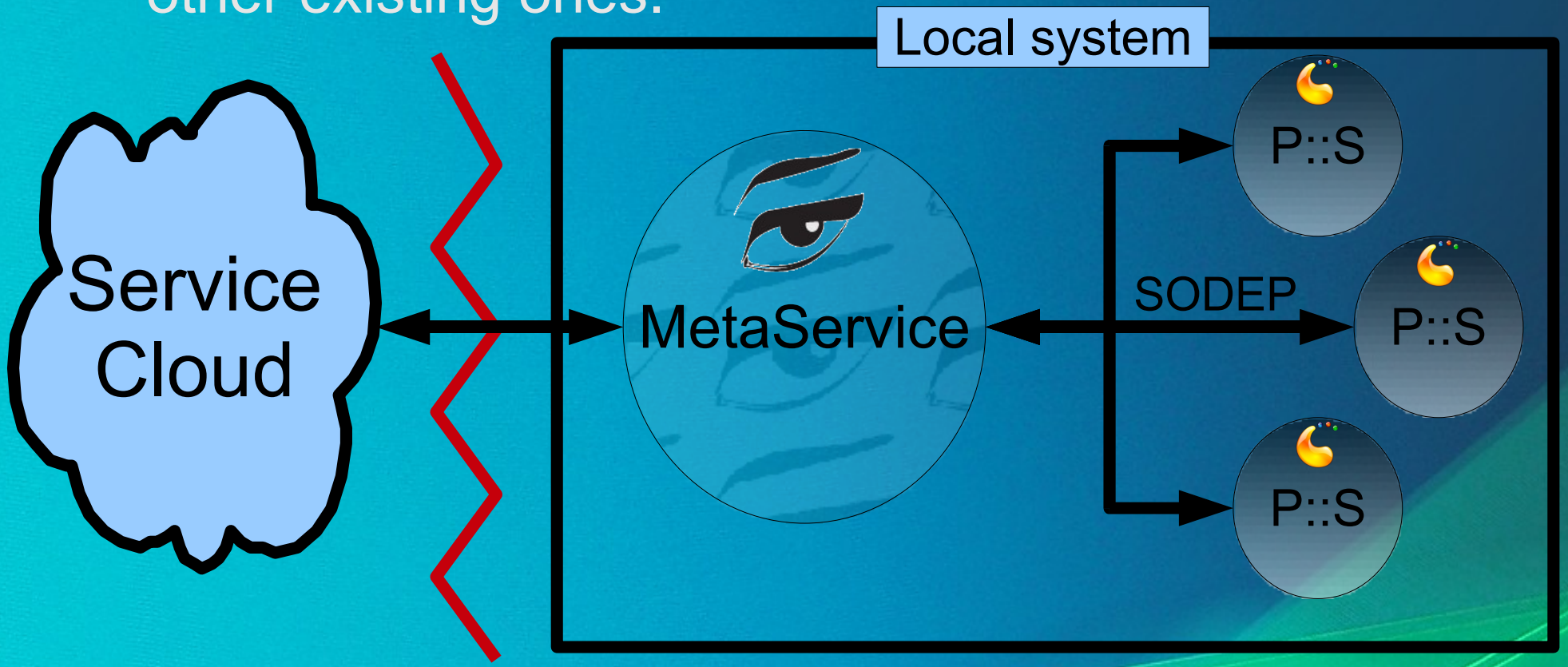
Plasma::Service

- Abstraction layer that allows Plasma applets to access and manipulate service data.



MetaService

- A single JOLIE service giving access to entire SOAs.
- Embeds other services and/or redirects requests to other existing ones.

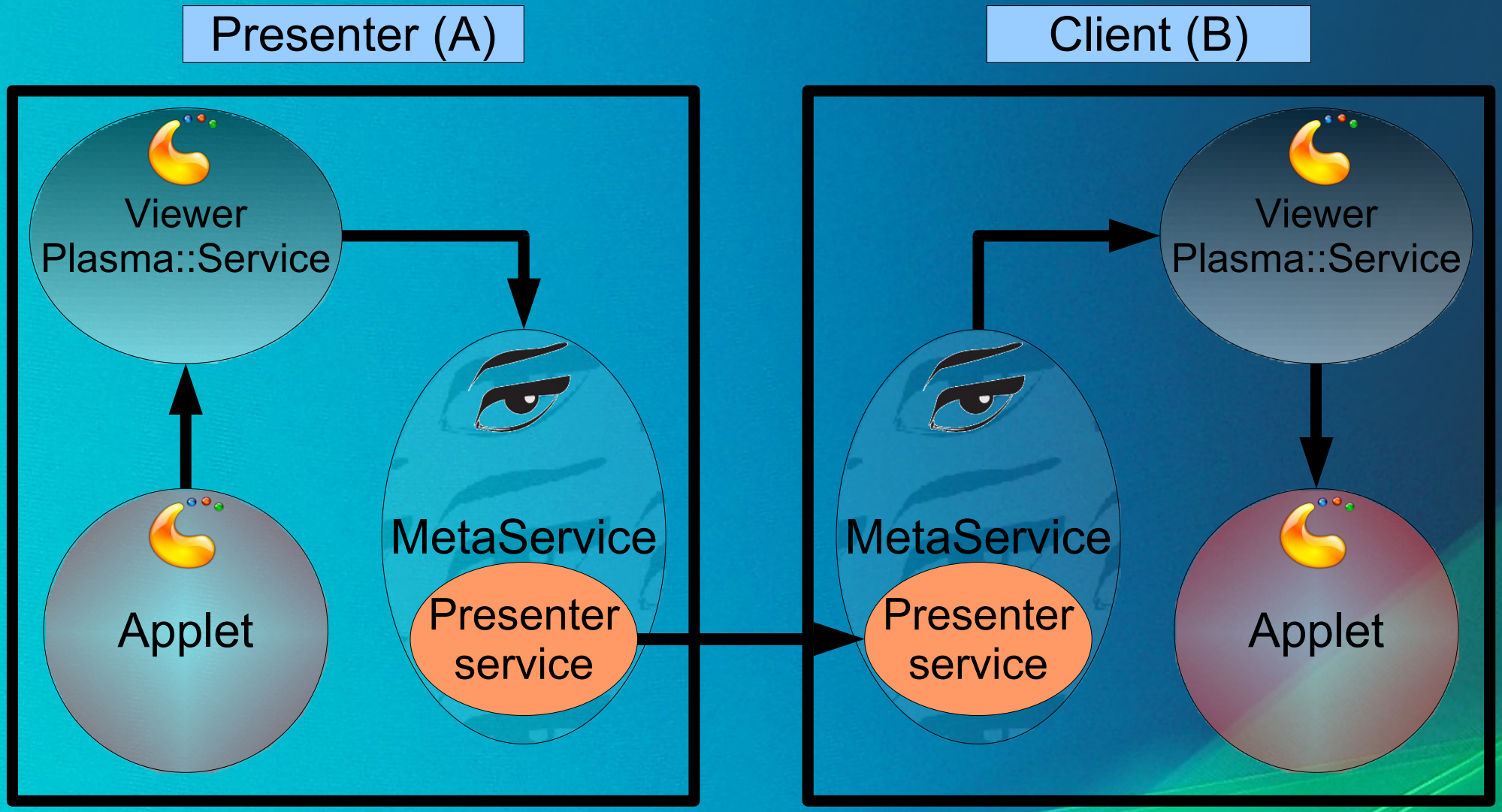


italianaSoftware

MetaService (2)

- MetaService features:
  - communicate seamlessly with existing services, local or remote;
  - deploy JOLIE services (by embedding), making them accessible by others;
  - unique communication endpoint (TCP/IP) for all of your services;
  - deploy Plasma data handlers as services (WIP).
- Plasma::Service allows plasmoids to make use of all of these features.

Example: Vision revisited



Conclusions

- The new architecture enables the seamless integration of UI components (applets) with services.
- Easy service access with the Plasma::Service API.
- Easy service writing with the JOLIE language and MetaService-based deployment.
- New possibilities for making applications that exploit service-oriented desktops.
- Service-oriented Computing brought to every desktop user and programmer.

Coming soon...

KDE 4.2

2009

Q&A

Do I actually have time left?

Thank you!