Building reflective mobile middleware framework on top of the OSGi platform

ICSR9, Torino

Gabor Paller, gabor.paller@nokia.com

2006.06.12

NOKIA
Connecting People

Table of contents

- Mobile middleware patterns
- OSGi R4 support of mobile middleware patterns
- Mobile middleware framework elements
- Results

Mobile middleware patterns

- Huge literature
- Two major modes of operations
  - nomadic (client-server with temporarily disconnected clients)
  - peer-to-peer (no distinguished node at all)
- Most important patterns start to emerge
  - Context-aware
  - Reflective
  - Off-line access
  - Asynchronous communication
- Often mentioned but **not part of this work**:
  - Deployment
  - Security (code security, policy)

Pattern: context-awareness

- Traditional middleware is strictly layered
  - it shields the applications from events concerning the lower level of the stack.
- Context-awareness -> no such shielding
  - the application is aware of the environment situation.
- Context changes are inherently asynchronous
  - often delivered in the form of events.
- Only the application can decide what context events are important and how to handle them.

Pattern: reflection

- Other side of the context-awareness story
- Reflection → the program is able to make computations on its own structure during its execution
  - (retrieve the current structure, evaluate the structure against environmental constraints then update the structure if necessary). Reflection is a crucial technique in mobile
- Context-aware applications -> large number of context state combinations.
- The middleware needs to be decomposed into a collection of smaller components.
  - The application chooses just the components it needs and composes the middleware that serves the application the best.
- In case of context changes, the application evaluates the context transition and possibly changes the instantiated components and/or their configuration (connection and component configuration).

Pattern: off-line access

- Disconnection is an inherent property of mobile computing.
  - The reason for disconnection can be physical (no coverage) or social (mobile access is too expensive or not acceptable in the given situation).
- Disconnected mode must be available.
  - Relocation of relevant data and code to the mobile device.
- Data relocation can be achieved by pretty established data synchronization techniques.

Pattern: asynchronous communication

- Networked computing is dominated by solutions following Remote Procedure Call (RPC) semantics.
  - the calling procedure is suspended for the duration of the call and execution continues in the called procedure. RPC is inherently synchronous.
- Mobile transport networks are characterized by long and variable delays and frequent transmission errors.
- Communication must be asynchronous
  - (event-based or message-based terms are also used for the same concept).
- Instead of procedure call semantics -> events are delivered to the application.

OSGi

- www.osgi.org
- 4th release available (R4)
- OSGi is a Java-based interface specification between management-aware applications and the management platform itself.
- Key principle of OSGi is asynchronous (”hot”) management which means that running applications must be prepared to react immediately to management interactions.
    - For example one software component may be upgraded to a newer version and applications using the component must be able to gracefully migrate from the old component to the new one.
- OSGi is a very dynamic environment.
- Applications and the platform are involved in extensive event communication and well-behaving OSGi application uses these events to coordinate with the platform in case of asynchronous management actions.

OSGi software stack

- **Security layer**: This layer specifies how to sign OSGi artifacts.
  - This layer builds heavily on Java JAR signing and the bundle concept (see module layer).
- **Module layer**: This layer is responsible for the management of the installable software modules called bundles in OSGi parlance.
  - A bundle is a collection of resources (Java class files and/or static files) and metadata that are added or removed by one operation.
  - Bundles can export and import Java packages creating a dependency web among bundles.
- **Lifecycle layer:** Bundles can be started and stopped, clearly demarcating the operational and maintenance mode of the bundles.
  - When a bundle is taken into use, it is often started.
  - Before management of the bundle begins, the bundle is stopped.
- **Service layer:** Services are key abstractions in OSGi.
  - An OSGi service is much reminescent to a component port: it is associated to a Java type (preferably interface type) and has meta information attached to it.
  - Services are registered in the service registry

OSGi components

- New in OSGi R4. SCR (Service Component Runtime) manages components.
- Influenced heavily by ObjectWeb's Fractal component model (fractal.objectweb.org)
- OSGi component: provided and required services described in declarative way (XML)
- Constraint language operates on service meta-information. Plus service multiplicity (e.g. how many consumers can be connected to a service) can be declared.
- SCR binds services automatically based on service constraints.
- Components have own lifecycle controlled by SCR.
- Component network can be manipulated programmatically by enabling/disabling components
- Components can also be created programmatically by means of component factories.

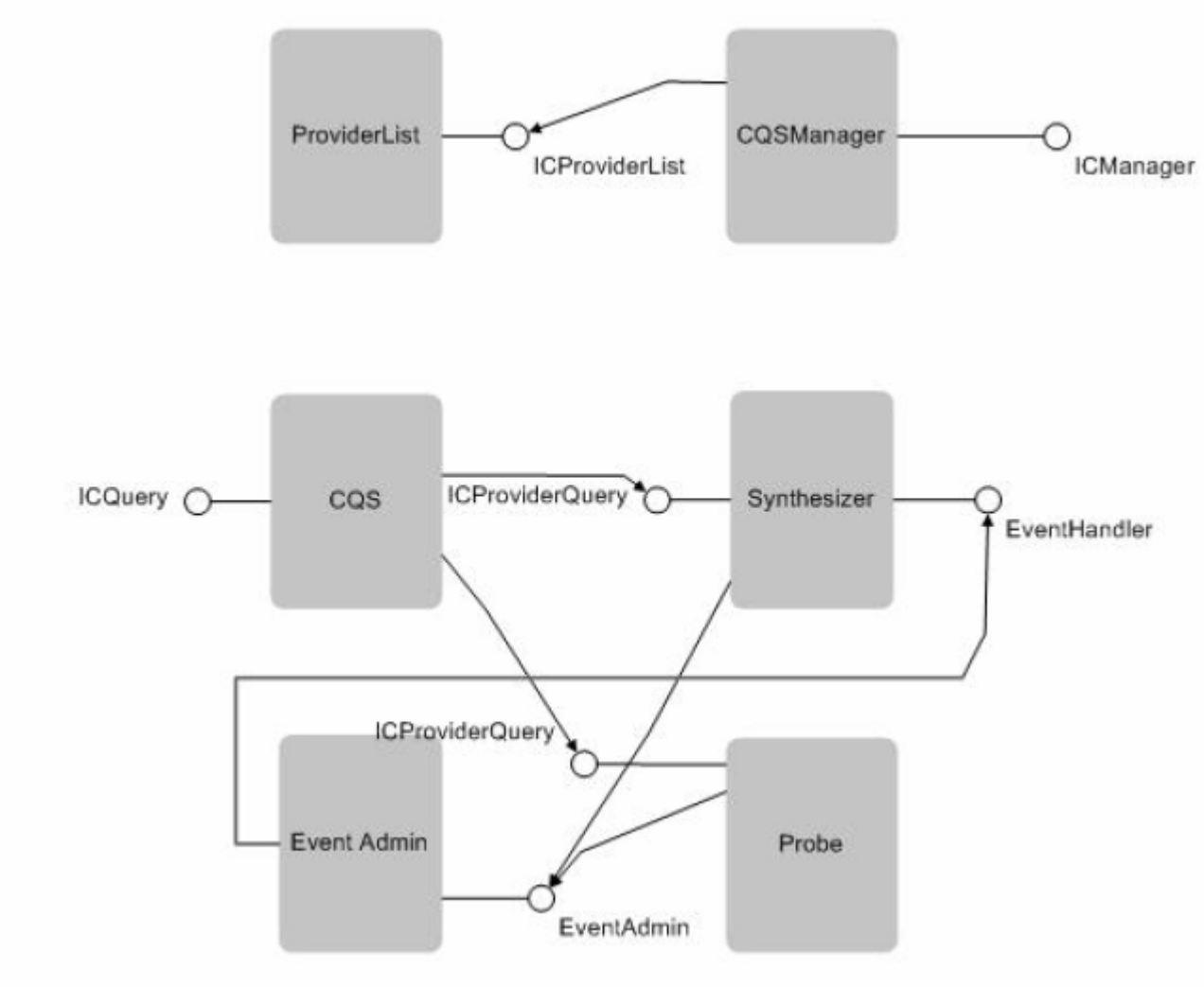
Reflection in our framework is built on OSGi components

- Meta-spaces: each application have their own space of middleware components. For example one application may use one specific bearer (e.g. WLAN), the other may use any bearer.
- Every component is generated by their component factories. This guarantees that separate component instances are created for each application metaspace.
- Every component is tagged by mwfw.app.id whose value is a string unique for each application. Components with the same mwfw.app.id property belong to the same meta-space.
- “Blackout” when reconnecting the component network. OSGi component space is not hierarchical!
- General problem with OSGi: separation. Everything runs in the same JVM, resource management is an issue if malfunctioning or hostile applications are assumed.
- One extra problem created by SCR: components created by component factories don’t belong to any application.

Context subsystem

- **Low-level context:** Obtaining environmental information from software or hardware sensors like network cards, file system monitors, etc.
- **High-level context:** For example the context element ”device from the same group nearby” can be generated from ”Bluetooth device in range” low-level context element if the Bluetooth device is recognized as a device belonging to some application defined group.
- Push (event) and pull (query) access

Context subsystem component network



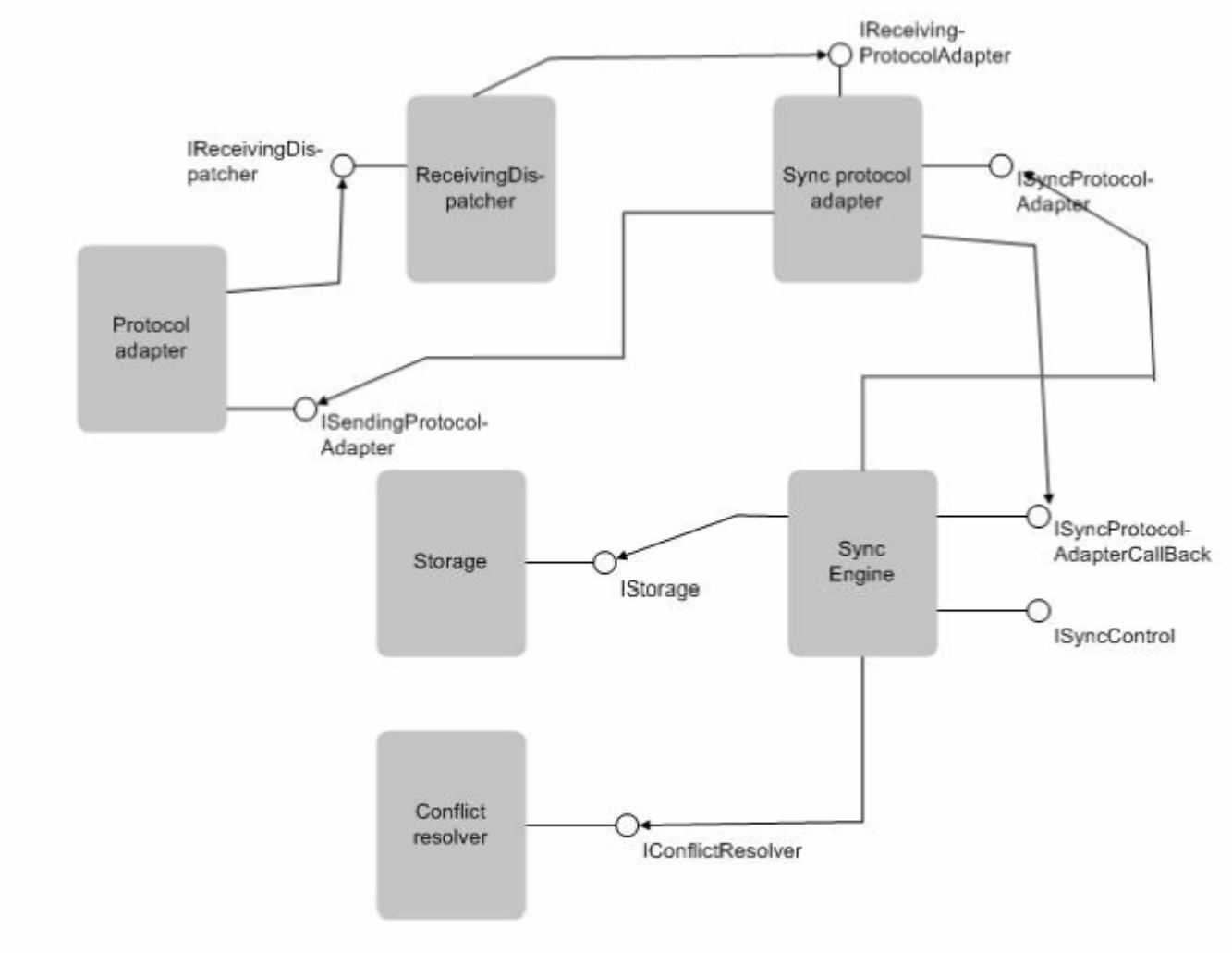
Context subsystem: highlights

- Uses OSGi Event Admin publish-subscribe service (new in R4) to deliver events to interested applications.
- Clumsy hacking with ProviderList component and CQSManager to enable/disable context providers because OSGi R4 has no central component directory

Off-line access pattern

- Although time-stamp-based synchronization is the most common technique, the solution must be able to incorporate alternative sync engines. Particularly, full backup/restore (slow sync) and CPISync were identified as interesting candidates for more special scenarios.
- The solution must be able to incorporate multiple sync protocols like SyncML DS, ActiveSync, etc.
- It must be possible to use any network bearer for the synchronization.
- In addition, it must be possible to change the bearer during a synchronization
- The storage abstraction must be flexible enough to incorporate databases, plain files and applications acting as data sources.
- Application must be able to provide their own conflict handling logic to resolve synchronization conflicts.

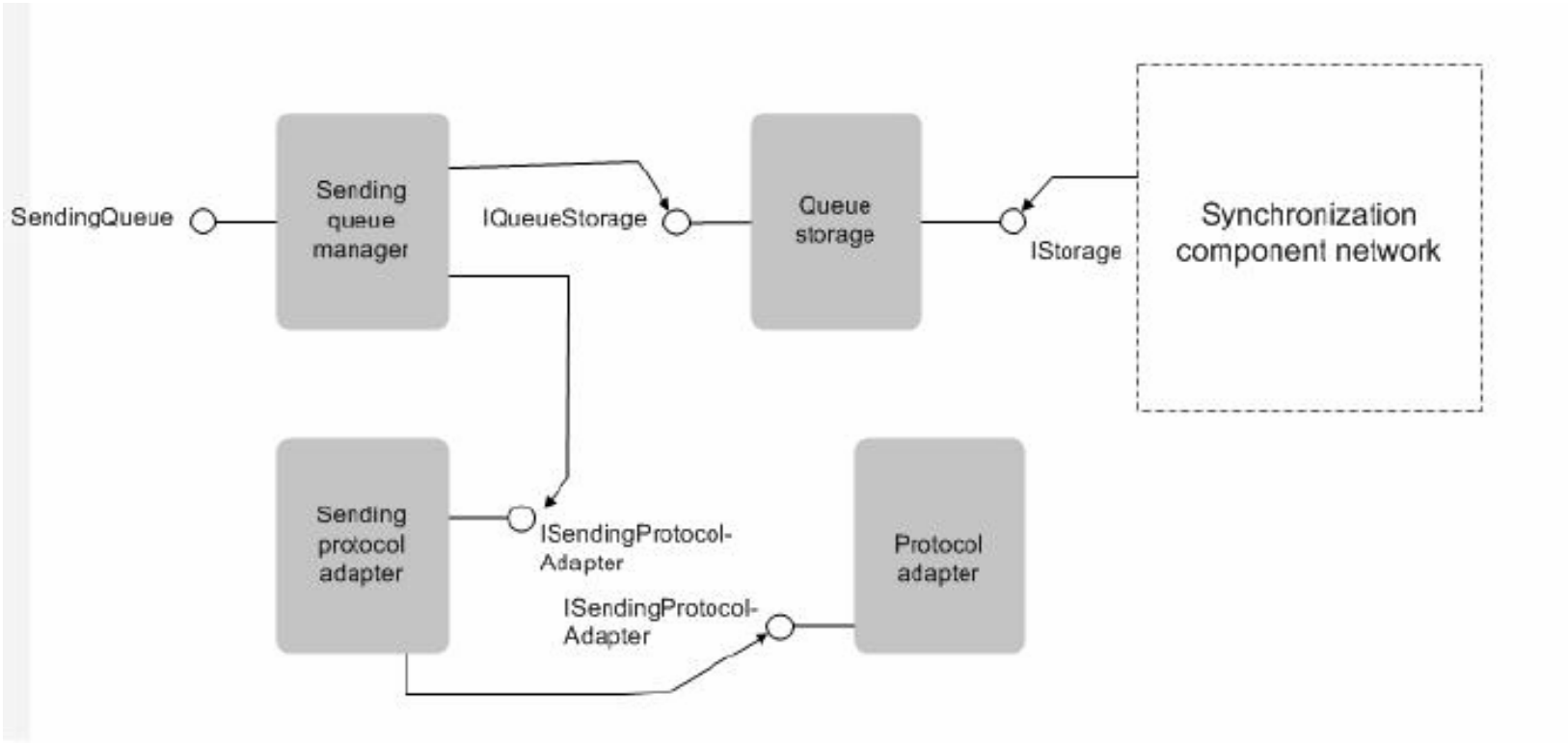
Synchronization: component network



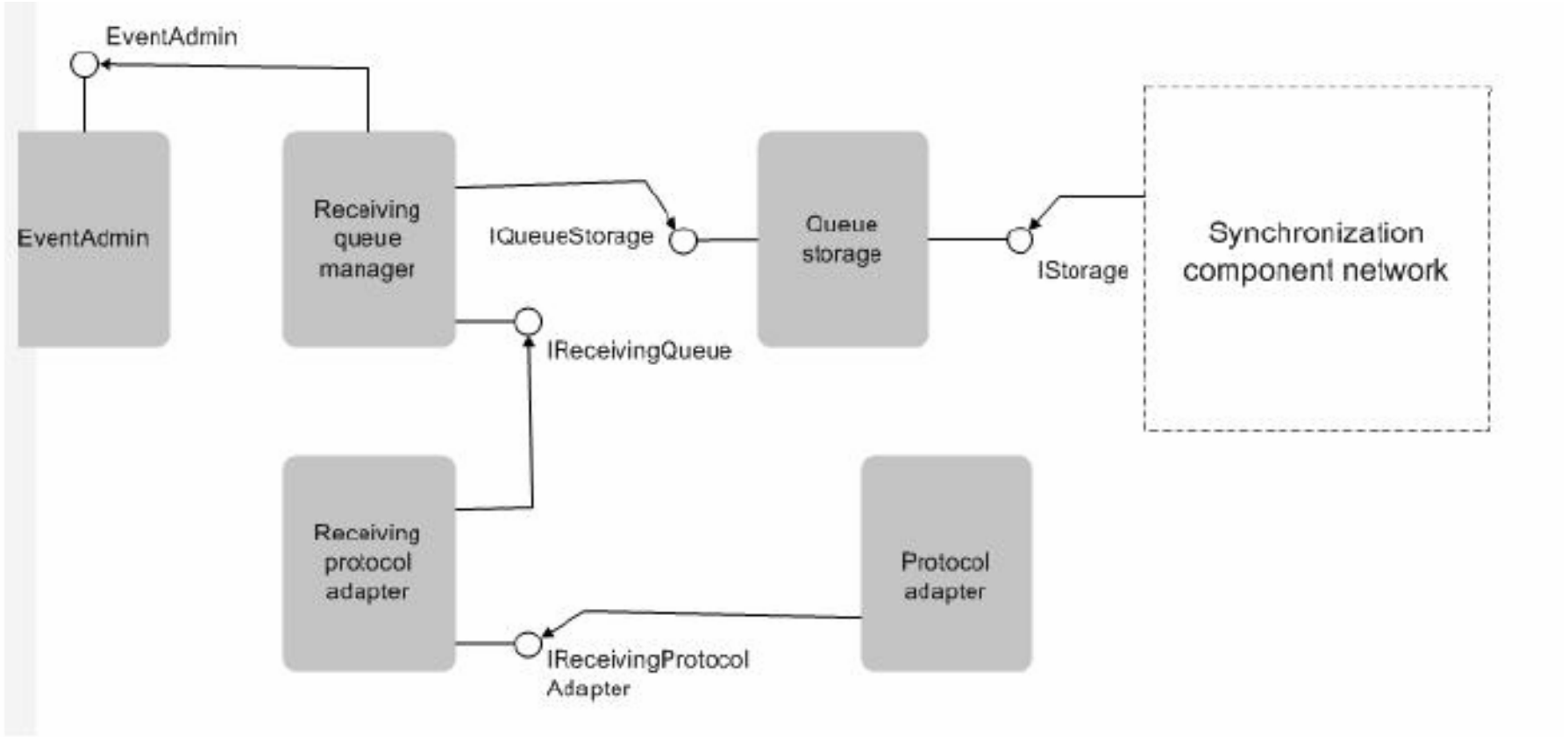
Asynchronous communication

- Asynchronous communication
  - Queues
    - Explicit addressing
    - Publish-subscribe
  - Tuple spaces
    - Essentially a distributed hashtable
    - Key-value pairs
    - Replication among nodes
- Each approach can be implemented by its own component network
- Only queue with explicit addressing was implemented in the prototype

Queue: component network (sender)



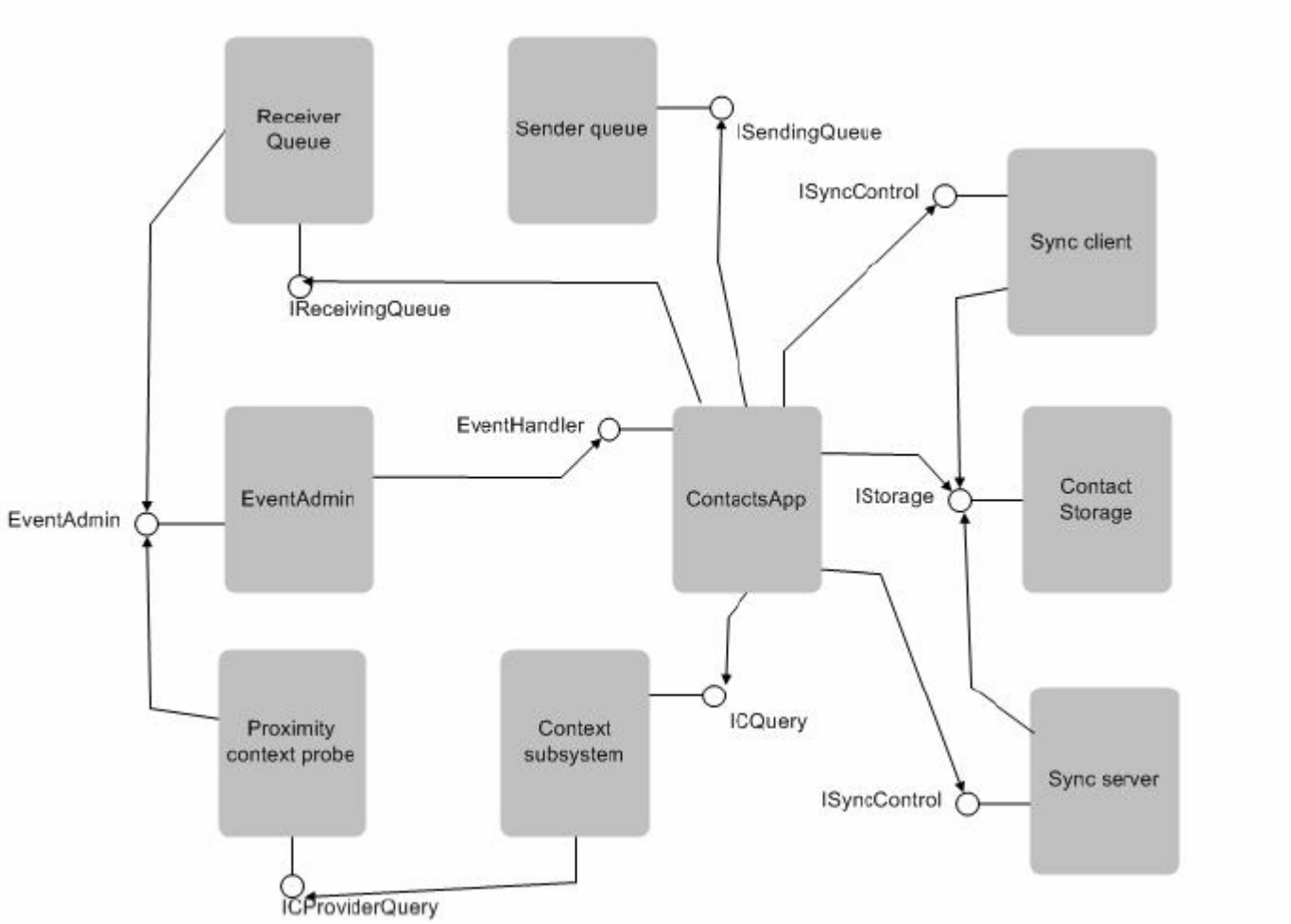
Queue: component network (receiver)



Queue: highlights

- Uses synchronization component network to enable queues to work in off-line mode. Dual or mixed modes are also available.
- Uses Event Admin to deliver queue entries on the receiving end

Sample application



Results

- Static sizes (uncompressed class file sizes):
  - Framework core+interfaces: 55231 bytes
  - SyncML DS sync implementation: 86857 bytes
  - Contact application, PC setup: 71748 bytes
  - 11.7 MBytes static footprint (OSGi core+OSGi standard services+framework core+demo app)
- Dynamic footprint after succesful synchronization (OSGi R4 reference implementation)
  - Linux, standard JRE 1.4.2, totalMemory/freeMemory
    - Sync initiator: 2567248 bytes
    - Sync server: 2691064
  - Nokia 9500, IBM J9
    - 5.8 Mbytes RAM footprint (JVM+OSGi+middleware framework+application)

Conclusions

- Reflective model is attractive in the mobile application space because of its low footprint and easy adaptability to changes in the environment.
- OSGi R4 provides a powerful component system which is made even more interesting by its dynamic component wiring capabilities.
- Dynamic wiring, if not used carefully, may cause “blackout” in the operation of the system.
- OSGi R4 as it is provides weak application separation.
  - There are many ways to exploit this protection system and malicious applications are able to disrupt other applications.
  - Our reflective middleware framework is built on component factories that complicate the separation problem further.
- Applications are harder to develop, debug and test in the dynamic component model supported by OSGi R4 than using the monolithic approach.
- OSGi R4-based applications have realistic footprint on PDA-class devices and smartphones.