Context Management

Sirenia — Oct 2020

The CCOW standard

CCOW defines a protocol which orchestrates *synchronisation and sharing of state* between applications on the same desktop.

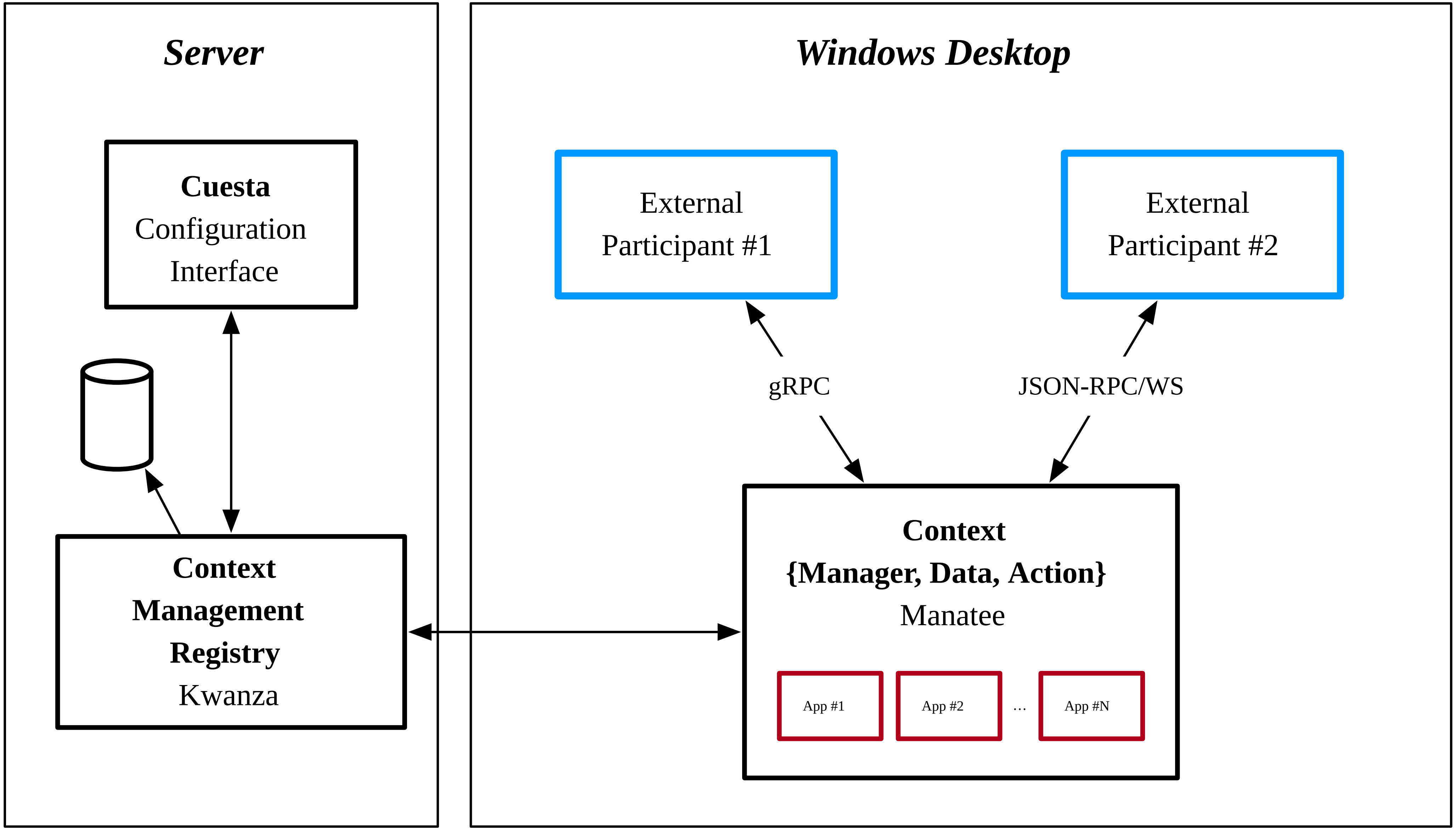
The shared state is called the *common context* and changes to it are executed transactionally.

All participants must subject themselves to the state of the common context *updating their internal state* to match it. Similarly they must update the common context once relevant internal state changes.

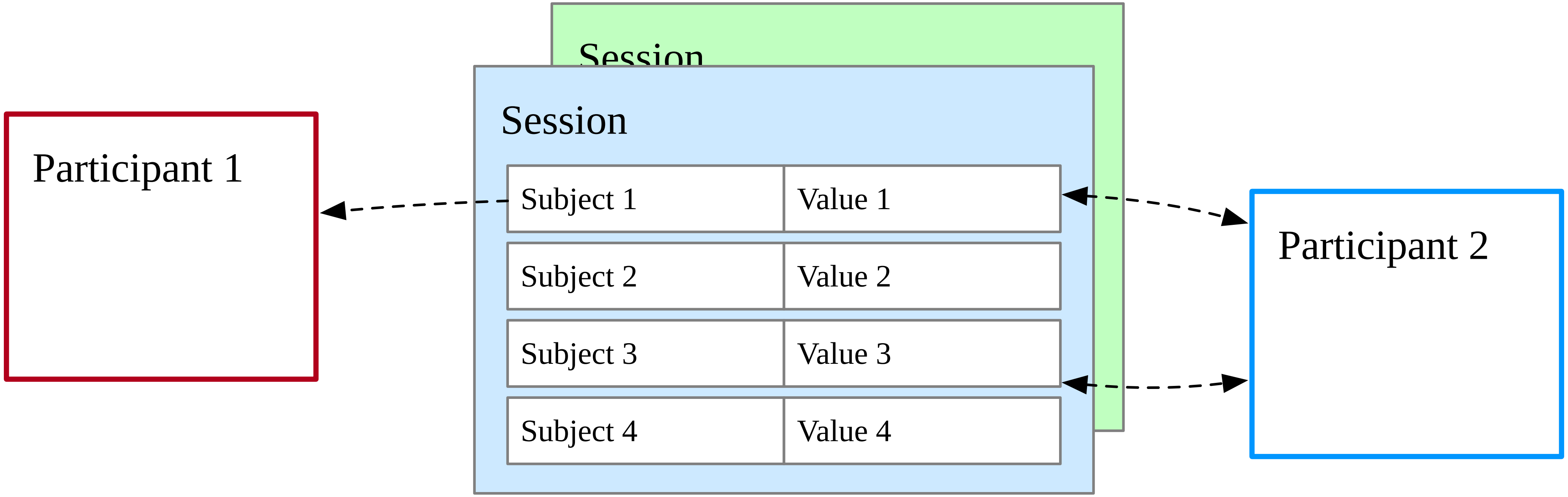
The central component in this interaction is a **context manager** (CM) which mediates changes and notifications to and from all participating applications.

Another important component is a **context management registry** (CMR) which is a key-value store containing configuration and run-time information.

Applications that interface with the context manager and join the common context are called **context participants**.


Server
Windows Desktop
Cuesta
Configuration
Interface
External
Participant #1
External
Participant #2
gRPC
JSON-RPC/WS
Context
Management
Registry
Kwanza
Context
{Manager, Data, Action}
Manatee
App #1
App #2
…
App #N

The common context



Subject 1	Value 1
Subject 2	Value 2
Subject 3	Value 3
Subject 4	Value 4

Our implementation of CCOW

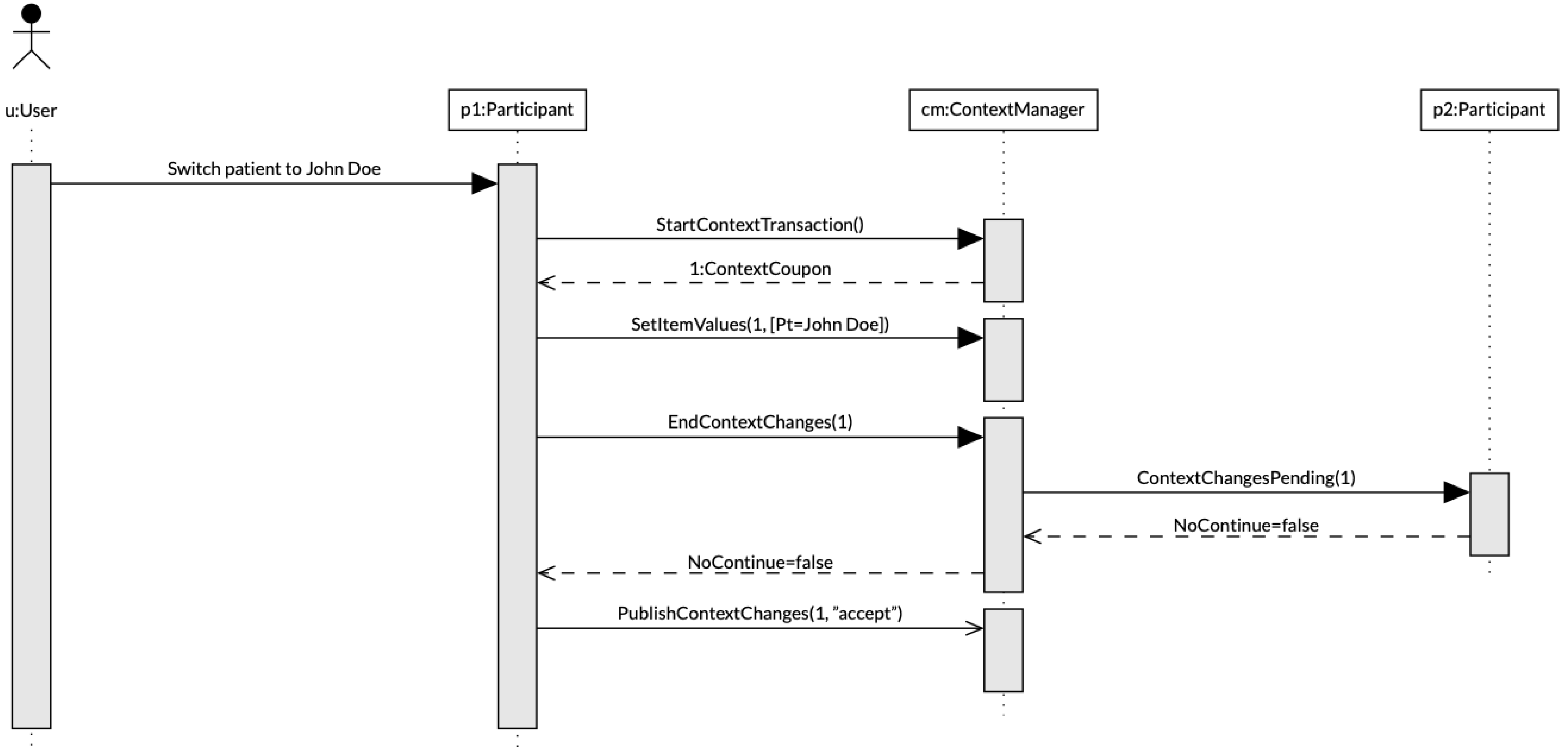
- *External* and *internal* participants.
  Internal participants are driven by a driver that matches their type and do not require use of an API. For legacy applications for instance.
- Automation framework built on top of the CM and using the drivers for supported frameworks.
  Allows for letting non-CCOW aware applications take part in context synchronisation.
- Contextual launching of automation actions (context actions).
- Export of information from the CMR to allow *external* participants to launch (automation) actions.
- Remote CM for cross-device, remote desktop and Citrix operation

External participants

- Applications which already support the CCOW standard
- The application has its own internal logic for context changes
- The application can signal that a context-change is not possible
- Application has its own UI to
  - indicate participation state and session adherence via colors
  - for letting users resolve conflicts
- Application may expose configured actions to allow users to run actions in other participants through the CM

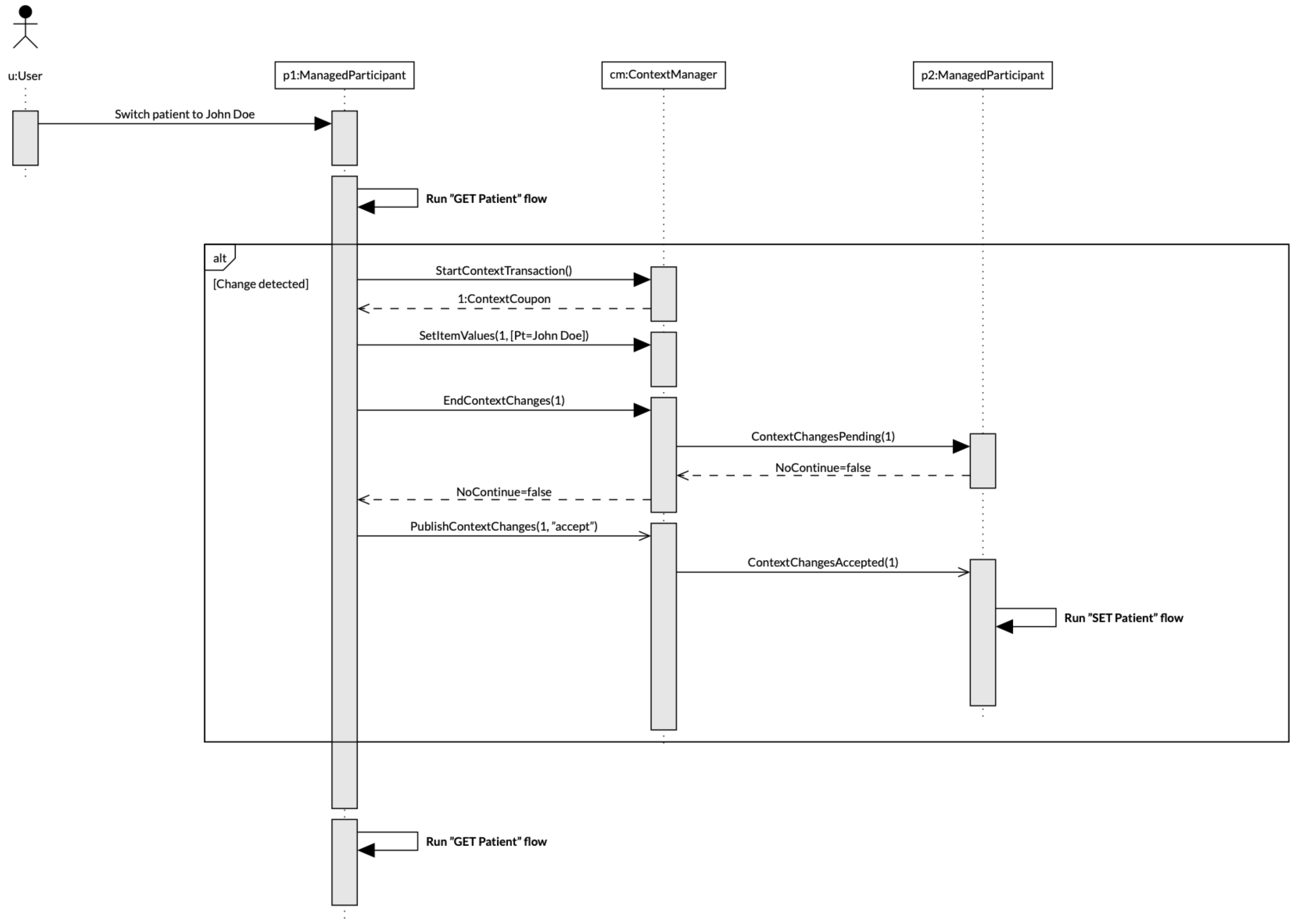
Requirements for external participants implementations

- **Must** follow common context state changes *or* leave the common context
- **Must** update common context on internal state change
- Configuration for each participant must be added to the CMR


u:User
p1:Participant
cm:ContextManager
p2:Participant
Switch patient to John Doe
StartContextTransaction()
1:ContextCoupon
SetItemValues(1, [Pt=John Doe])
EndContextChanges(1)
ContextChangesPending(1)
NoContinue=false
NoContinue=false
PublishContextChanges(1, ”accept”)

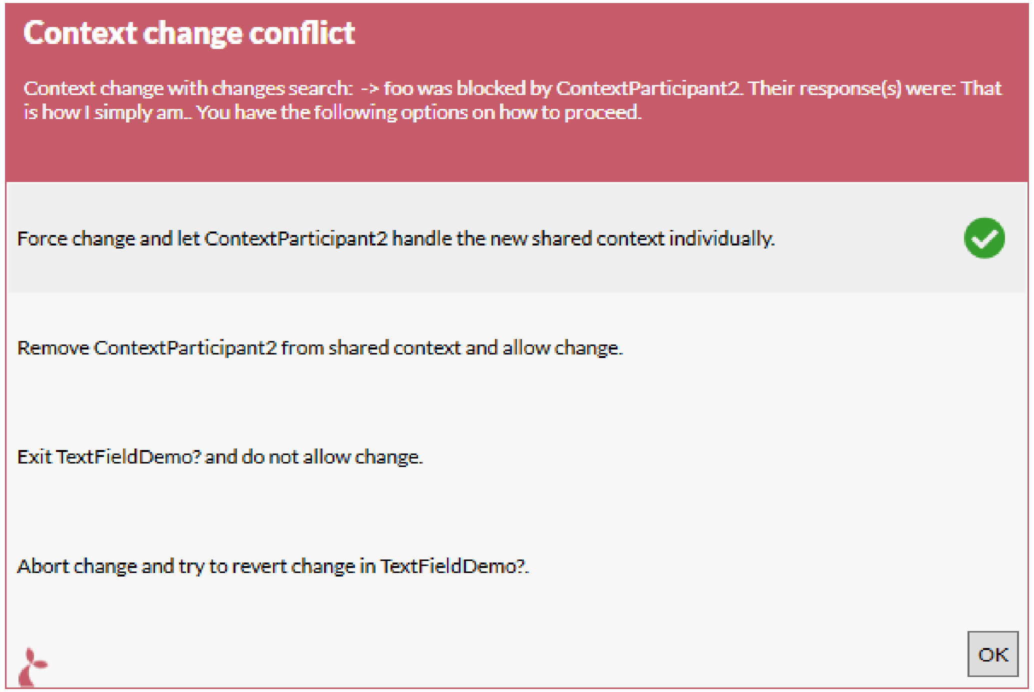
Internal participants

- Applications that does not support the CCOW standard can be made into internal participants via automation
- We’ll inject a driver layer between the Context Manager (CM) and the application which translates between CCOW operations and automation tasks
- Uses state-flows to specify how to read and write state from the application to be synchronised with the shared state
- No changes needed in the original application
- *Mapping agents* may be implemented as headless internal participants


u:User
p1:ManagedParticipant
cm:ContextManager
p2:ManagedParticipant
Switch patient to John Doe
Run "GET Patient" flow
alt
[Change detected]
StartContextTransaction()
1:ContextCoupon
SetItemValues(1, [Pt=John Doe])
EndContextChanges(1)
ContextChangesPending(1)
NoContinue=false
NoContinue=false
PublishContextChanges(1, "accept")
ContextChangesAccepted(1)
Run "SET Patient" flow
Run "GET Patient" flow

Conflict handling with *external participants*

If a situation arises in which an external application cannot follow the changes it must report `NoContinue = true` in the context transaction and the user must ultimately the decide how to proceed.



Context change conflict

Context change with changes search: -> foo was blocked by ContextParticipant2. Their response(s) were: That is how I simply am.. You have the following options on how to proceed.

Force change and let ContextParticipant2 handle the new shared context individually. ✓

Remove ContextParticipant2 from shared context and allow change.

Exit TextFieldDemo? and do not allow change.

Abort change and try to revert change in TextFieldDemo?.

Change detection for *internal participants*

A change in an internal participant is detected via continuous polling of its state.

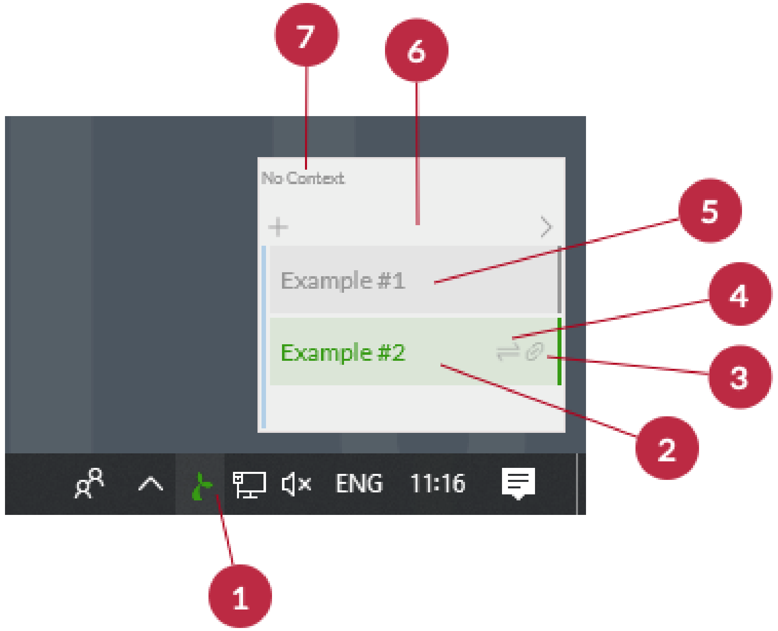
Once a change is detected a normal context transaction is initiated to update the shared context.

The polling frequence is controlled via the `ChangeDetectionDelayInMs` setting.

User interface elements

Menus and list of particpants

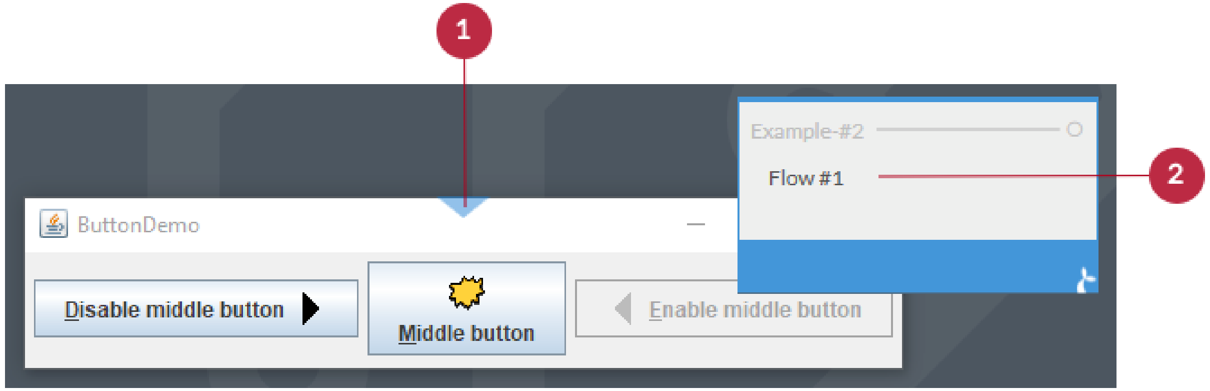
1. Manatee icon - shows participant-list on click and context menu on right-click.
2. Application is a participant and synchronised with the shared state (green), the color of the application can otherwise indicate:
    - *Purple* means the application is not yet ready to synchronise its state.
    - *Red* means the application state is "dirty", i.e. does not match the shared state.
    - *Blue* indicates the application has not yet synchronised with the shared state.
3. Handles for extricating the application from the shared state.
4. Indicates whether the application supports reading/writing/both of its internal state (some applications may only support setting the state if the application is not focused).
5. The color *gray* indicates that the application is not a participant.
6. Click to see all shared contexts (>) or (+) to create a new shared context.
7. Shows the value of a selected subject in the shared state.



User interface elements

The colors of common contexts

A session will automatically get a color when it is created. This color will be shown in applications (e.g. at (1)) and used in the action-launcher menu.



Sessions are automatically created if more than one instance of an application tries to join a context. There can only be one of each type of application in a context.

Technology mappings

Communication with external participants are supported with the following technology mappings. Note that *two-way communication* is needed.

- PROTOBUF/gRPC
- JSON-RPC/
  - WS(S)
  - NamedPipes
  - Netstring/TCP
- JSON/HTTP
- URLENCODED/HTTP
- JSON/HTTP(WS)

PROTOBUF/gRPC tech mapping

- HTTP2 used as transport.
- API is specified in a `.proto` file containing message types and service definitions.
- Support for bi-directional streams.
- Encryption built in.
- Fast and scales.
- Fairly heavy-weight in some implementations □
- Stubs are generated for many languages - more info at grpc.io.

JSON-RPC tech mapping

- A simple, lightweight RPC protocol with JSON encoded messages.
- Request/reply-, notification- and error type messages.
- Transport protocol agnostic, we support:
  - WebSockets
  - NamedPipes
  - Netstring/TCP
- More info at jsonrpc.org.

HTTP tech mapping

- Both REST- and RPC-style interaction supported
- Encryption support requires manual configuration
- ContextParticipants must expose HTTP server for callbacks □
- *Not recommended for use* except where Manatee is running on a server

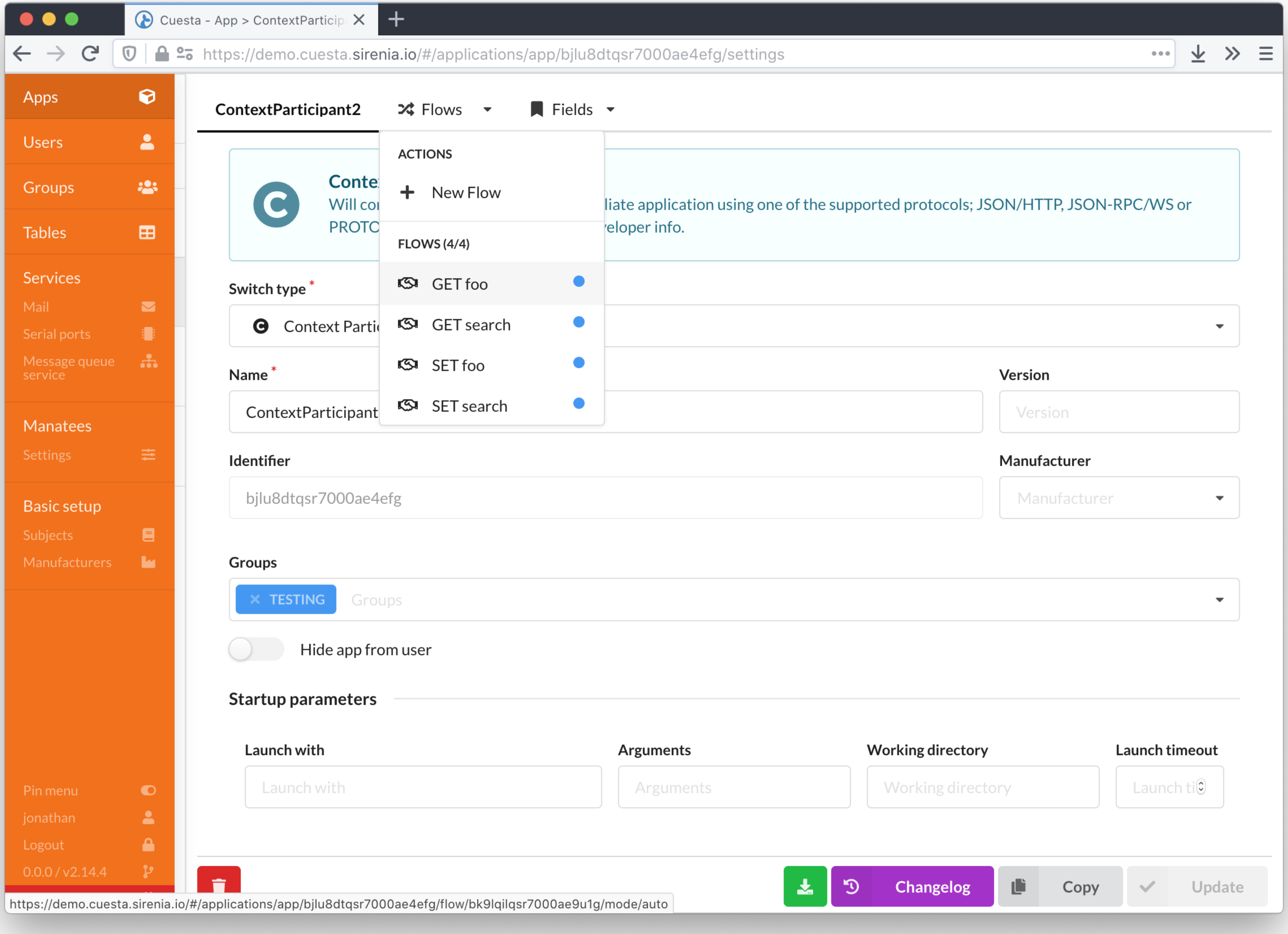
Setup

The `CMR` needs to know the capabilities of all participants including external context participants.

Register the info via `Cuesta` or use the `Kwanza` API to register it.

Information needed is;

- Application details; *name* etc – *id* is needed to identify to the CM
- Which *groups* should the application be made available to
  - Groups determine which users/machines etc has access to the application
- Which *subjects* are supported by the application
  - Read/write for each subject



Conte
Will co liate application using one of the supported protocols; JSON/HTTP, JSON-RPC/WS or PROTO veloper info.

Switch type *: Context Parti

Name *: ContextParticipant

Version: Version

Identifier: bjlu8dtqsr7000ae4efg

Manufacturer: Manufacturer

Groups: × TESTING Groups

Hide app from user

Startup parameters

Launch with	Arguments	Working directory	Launch timeout
Launch with	Arguments	Working directory	Launch ti

https://demo.cuesta.sirenia.io/#/applications/app/bjlu8dtqsr7000ae4efg/flow/bk9lqilqsr7000ae9u1g/mode/auto

API modus operandi

Follows the CCOW `ContextManager`, `ContextData`, `ContextAction` and `ContextParticipant` interfaces

1. Join the common context.
2. Implement the `ContextParticipant` interface and react to changes and action-requests.
3. Use the `ContextManager` and `ContextData` interfaces to initiate changes in the common context and to fetch initial context when joining.
4. Use the `ContextAction` to run actions in other participants.
5. Leave the common context.

Example source code and other links

Documentation for the JSON-RPC or gRPC mapping can be found at

- https://docs.sirenia.io/manatee/v1.29/external-apis/json-rpc/
- https://docs.sirenia.io/manatee/v1.29/external-apis/grpc/

Annotated (Java) example code (and source) for a gRPC participant can be found at:

- https://sirenia.gitlab.io/open/javacontextparticipant/src/main/java/JavaContextParticipant/RootCmd.html
- https://gitlab.com/sirenia/open/javacontextparticipant

These slides

- https://slides.sirenia.io/context-management
- https://slides.sirenia.io/context-management/slides.pdf