COSMOS: COntext entitieS coMpositiOn and Sharing

D. Conan and S. Leriche

Kick-off BROCCOLI

February 2008

Motivations and objectives

- Ubiquitous computing $\Longrightarrow$ High number of heterogeneous devices, huge amount of context data
- Context management [Coutaz et al., 2005] to identify/detect the situations of adaptations
- Process context data in a usable, scalable, and efficient manner
  - ♦ Usable: Compose, deploy, configure, and reconfigure (without programming)
  - ♦ Scalable:
    - ► No performance degradation when multiple clients' observations
    - + Separation of context collections according to context sources
  - ♦ Efficient: Control resources consumption of context management tasks (memory and activities)

Outline

1 Functionalities of a context manager 4

2 COSMOS concepts: Context node, context report, and context policy 5

3 Towards a case study: Mobile commerce 9

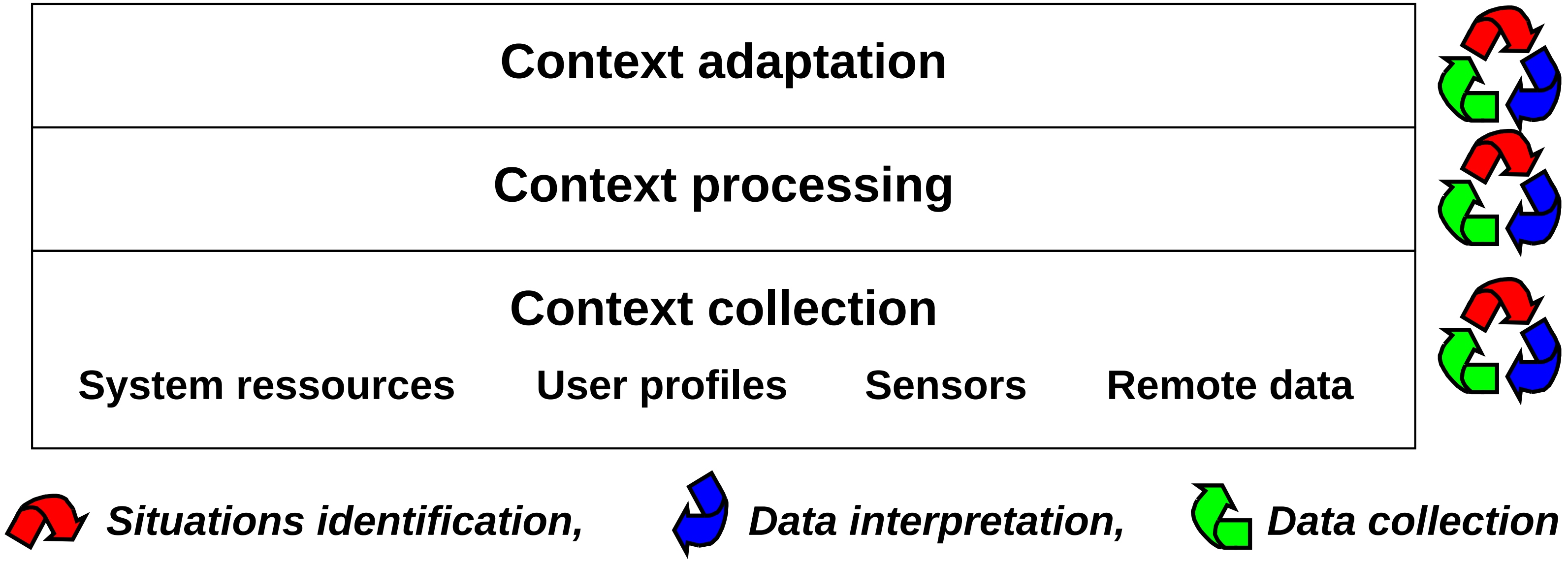
4 Status of the COSMOS framework 11

5 Ongoing and future work specific to COSMSOS 12

6 Tentative agenda for the forthcoming months specific to COSMOS 16

1 Functionalities of a context manager

- Separation of concerns
  - Collection = different context sources
  - Interpretation = different inference engines
  - Adaptation = several “client” applications with different situation identifications
- Compose context frameworks in a component-oriented architecture



Context adaptation

Context processing

Context collection

System ressources   User profiles   Sensors   Remote data

Situations identification, *Data interpretation,* *Data collection*

2 COSMOS concepts: Context node, context report, and context policy

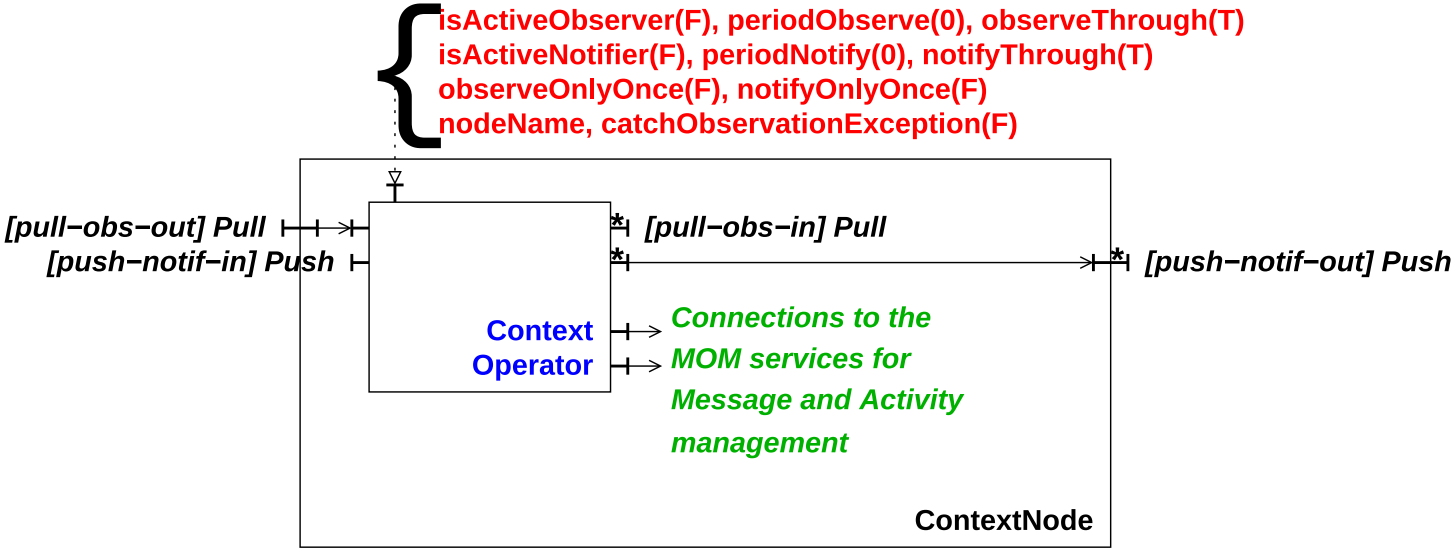
- Context policy = Abstract context information provided to the user/application
  - A hierarchy of context nodes
  - With sharing of context nodes between context policies
- Context node = Context information modelled by a component
  - Basic structuring “component” of COSMOS
- Context report = Extensible message structure
  - $[O..n]$ chunks: Identifier + values
  - $[O..m]$ sub-messages: Encapsulation

2.1 Software architecture approach: Context policies

- Apply architecture-based principles to design context policies
  - Software architecture for system instrumentation, deployment, configuration
    - "A software architecture of a program or computing system is the structure or structures of the system, which comprise software components, the externally visible properties of those components, and the relationships among them." [Bass et al., 1998]
- Use an architecture description language [Medvidovic and Taylor, 2000] to describe the context policy
  - Compose rather than program...
    - Reify a context policy as a tree of components with sharing
    - Architectural patterns for context node composition and sharing
  - ...during design, implementation, and execution
- Use a component-based message-oriented middleware [Leclercq et al., 2005]
  - Fine-grained management of context activities and context reports

2.2 Software component approach: Context node

- Apply component-based principles to design context nodes
  - Units for system modularity, reconfiguration, fault isolation
    - “A component is a unit of composition with contractually specified interfaces and context dependencies only. A software component can be deployed independently and is subject to composition by third parties.” [Szyperski, 2002]
- Compose rather than program...
  - When programming, apply attribute/annotation-oriented programming
  - ...during design, implementation, and execution


isActiveObserver(F), periodObserve(0), observeThrough(T)
isActiveNotifier(F), periodNotify(0), notifyThrough(T)
observeOnlyOnce(F), notifyOnlyOnce(F)
nodeName, catchObservationException(F)
[pull−obs−out] Pull
[push−notif−in] Push
*
[pull−obs−in] Pull
*
*
[push−notif−out] Push
Context
Operator
Connections to the
MOM services for
Message and Activity
management
ContextNode

2.3 Context node parametrisation

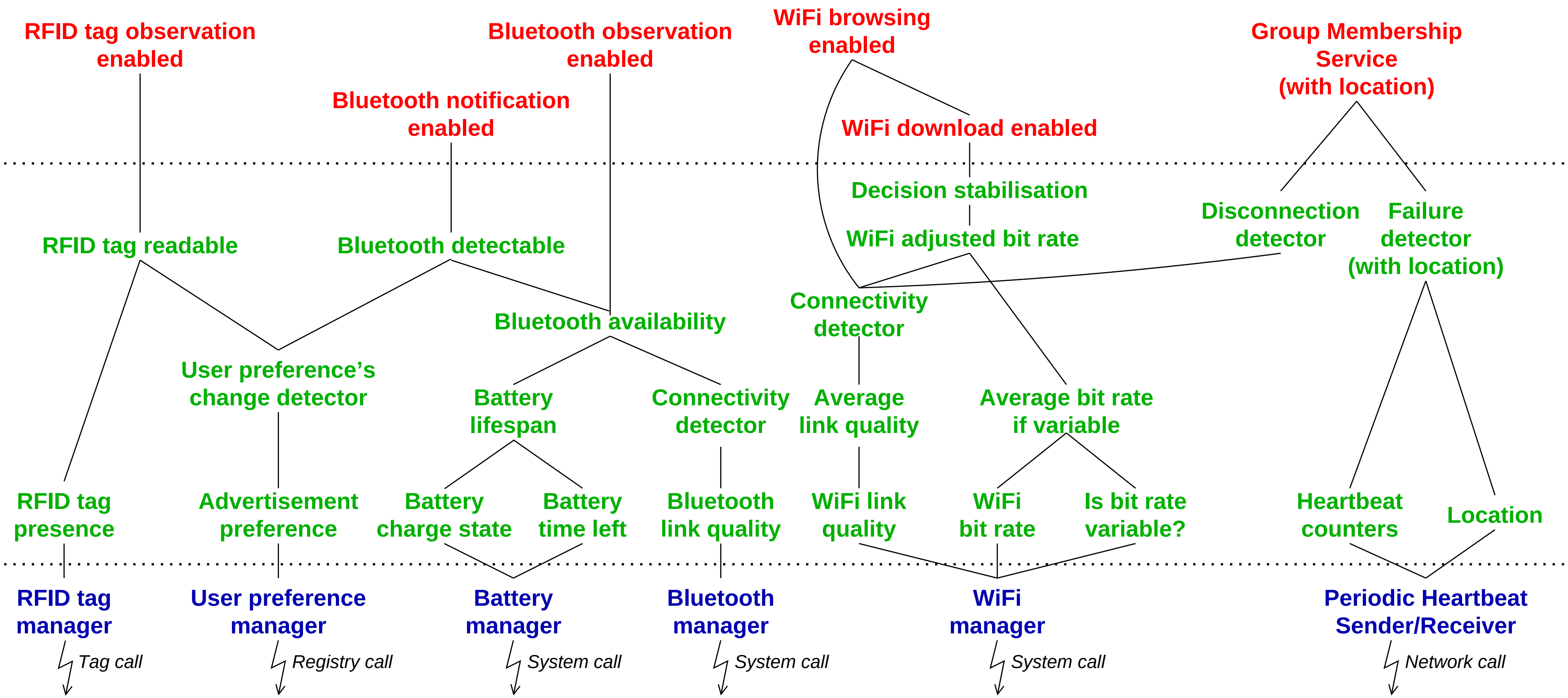
- Properties of a context node
  - Controls propagation of information
    - Can observe (down to the leafs) and/or notify (up to the root)
      - Attributes *Observe* and *Notify*
    - May block the context flow (down or up) or not
      - Attributes ObserveThrough and NotifyThrough
        + attributes ObserveOnlyOnce and NotifyOnlyOnce
  - Controls the propagation mode
    - Is passive or active
      - Attributes isActiveObserver and isActiveNotifier
        + attributes period*
  - Has a name to be registered into a registry and searched for for configuration
    - Attribute nodeName

3 Towards a case study: Mobile commerce

- Family shopping in a mall with all the members of the family equipped with a mobile device
    - Share information
    - Consult product prices
    - Download discount tickets
    - Be notified of advertisements
    - Access additional information and comments about a product
    - Find the location of a product or a shop in the mall

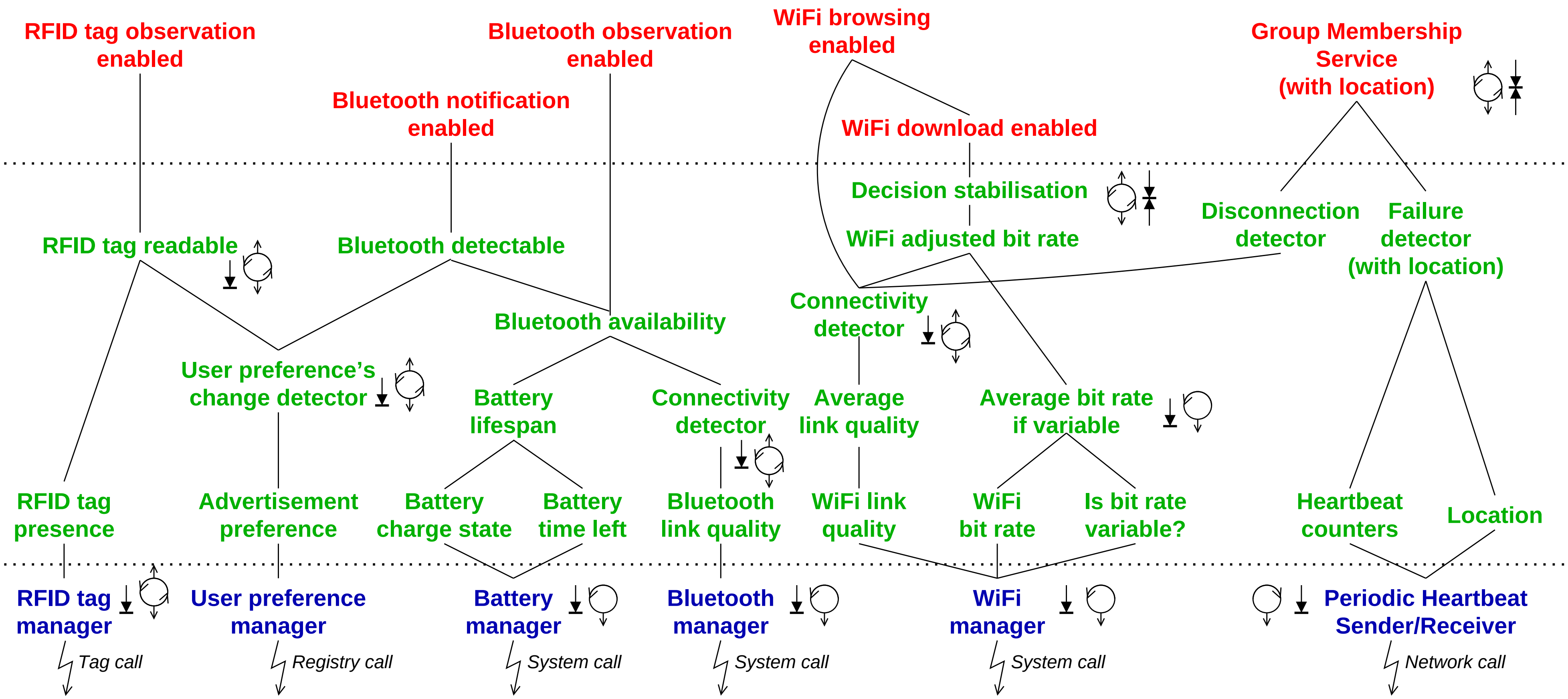
3.1 Context management with COSMOS

- Forest of context policies



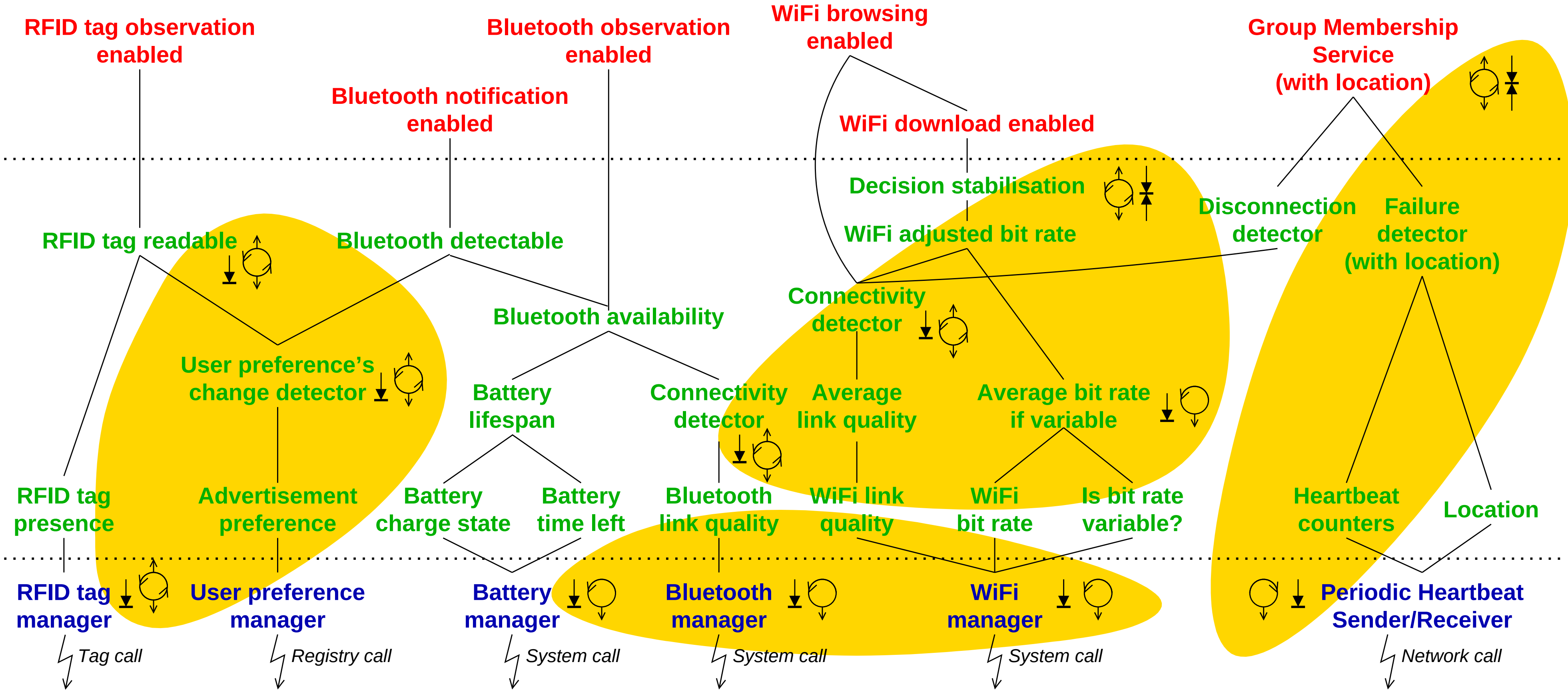
3.1 Context management with COSMOS

- Configuring context nodes: non-/blocking, active/passive



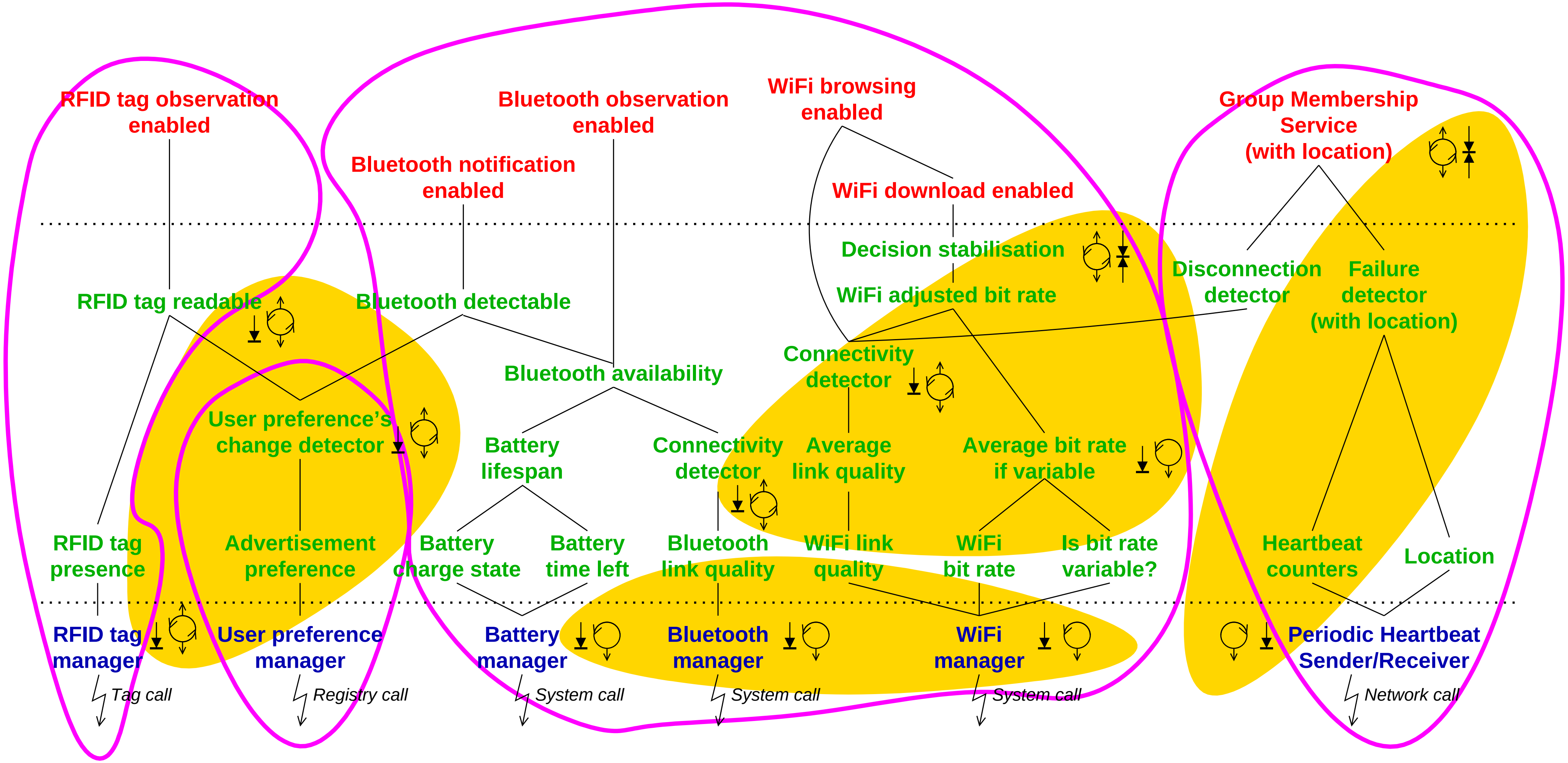
3.1 Context management with COSMOS

- Mapping context node activities to threads



3.1 Context management with COSMOS

- Mapping context nodes to message managers



4 Status of the COSMOS framework

- Publications: [Conan et al., 2007, Conan et al., 2008]
- Web site: `http://picoforge.int-evry.fr/projects/cosmos`
- Forge: `https://picoforge.int-evry.fr, guest/guest`
  - Project currently unstable, under a refactoring and mavenisation process
    - From Ant to Maven
      - Decomposition into `cosmoscore`, `cosmoslib` and `cosmossaje`
      - No deployment web site for the moment
    - From Fralet-Xdoclet to Fraclet-Java and Dream-Annotation
      - Dream-Annotation depends on Fraclet-annotation: No `@Legacy`
      - Perhaps conflicts between Fraclet-Java and Dream-Annotation
    - Unitary tests to replace `cosmossaje` tests
      - Especially, Dream activity management
    - Design pattern “Singleton” using dynamic sharing of components
      - Two many layers of composition $\Longrightarrow$ up to now, using “Singleton” objects
      - See email on the Fractal mailing list of Romain Rouvoy dated ...

5 Ongoing and future work specific to COSMSOS

5.1 Domain specific language for context composition

- DSL [Mernik et al., 2005] for writing context policies
  - Generate context policies written in Fractal ADL
    - First step towards analysis, verification, optimisation, transformation, etc.
      - *E.g.*, merge context policies, deadlock prevention

```
# Functional part: Compose context nodes
  sensor RfidTagMgr = RFIDTagRM[BO,AO,AN];
  sensor PrefMgr = UserPreferenceRM;
  processor AdvertisementChange = ChangeDetectorCO[BO,AO,AN]
                                      (PrefMgr.extract("advertisement-preference-chunk"));
  processor TagReadable = TagReadableCO[BO,AO,AN]
                           (RfidTagMgr.extract("tag-presence-chunk"),AdvertisementChange);
  processor TagObservationEnabled = IsEnabledCO(TagReadable);
# Extra-functional part: Threads and memory consumption
  task RFIDTask = AdvertisementChange,TagReadable,RfidTagMgr;
  thread RFIDThread = RFIDTasks[5000];
  reporting UserPrefReport = AdvertisementChange/descendant-or-self::*;
  reporting RFIDReport = TagObservationEnabled,TagReadable,RfidTagMgr;
```

5.2 Generic context operators

- Using Fractal-Generics
  - See email on the Fractal mailing of Philippe Merle dated Jan 16 11:23:25 2008

- First ideas

`processor Foo1 = add(BarInt1,BarInt2,1)`	Inputs/Outputs = Java primitive types
`processor Foo2 = add(BarInt1)`	Not the same number of Inputs
`processor Foo3 = and(BarBool)`	Not the same operator
`processor Foo4 = myOperator(Bar)`	Application-specific operator and chunk

  - Using a generic context operator
    - Argument = Method of the operator (*e.g.*, `cosmos.op.add` or `myapp.myOperator`)
    - Undefined number of child context nodes
  - Chunk types automatically deduced
    - Either a Java primtive type (*e.g.*, `cosmos.NumberChunk` containing a `j.l.Number`)
    - Or `dream.msg.AbstractChunk` returned by application-specific operators (*e.g.*, `myapp.BarChunk` containing a `myapp.Bar`)

5.3 Deployment and distribution of context information

- Deployment of COSMOS with FDF [Flissi et al., 2008]
  - Description of Dream software
  - Description of COSMOS software
- Distribution of context information with Dream [Leclercq et al., 2005]
  - Study of the Dream communication components library
- COSMOS as a network-accessible service
  - Dynamic instanciation/removal of new context policies
  - Dynamic merging of context policies
  - ? COSMOS = a distributed service

6 Tentative agenda for the forthcoming months specific to COSMOS

- End of March: Stabilisation of `cosmoscore`
- End of April: Generic context operators in `cosmoslib`
- End of May: First proposition of the DSL for context composition

References

[Bass et al., 1998] Bass, L., Clements, P., and Kazman, R. (1998). *Software architecture in practice*. Addison-Wesley.

[Conan et al., 2007] Conan, D., Rouvoy, R., and Seinturier, L. (2007). Scalable Processing of Context Information with COSMOS. In Indulska, J. and Raymonds, K., editors, *Proc. 6th*, volume 4531 of *Lecture Notes in Computer Science*, pages 210–224, Paphos, Cyprus. Springer-Verlag.

[Conan et al., 2008] Conan, D., Rouvoy, R., and Seinturier, L. (2008). COSMOS: composition de nœuds de contexte. *Technique et Science Informatiques*. À paraître.

[Coutaz et al., 2005] Coutaz, J., Crowley, J., Dobson, S., and Garlan, D. (2005). The disappearing computer: Context is Key. *Communications of the ACM*, 48(3):49–53.

[Flissi et al., 2008] Flissi, A., Dubus, J., Dolet, N., and Merle, P. (2008). Deploying on the Grid with DeployWare. In *Proc. 8th IEEE International Symposium on Cluster Computing and the Grid*, Lyon (France).

[Leclercq et al., 2005] Leclercq, M., Quéma, V., and Stefani, J.-B. (2005). DREAM: a Component Framework for the Construction of Resource-Aware, Configurable MOMs. *IEEE Distributed Systems Online*, 6(9).

[Medvidovic and Taylor, 2000] Medvidovic, N. and Taylor, R. (2000). A classification and comparison framework for software architecture description languages. *IEEE Transactions on Software Engineering*, 26.

[Mernik et al., 2005] Mernik, M., Heering, J., and Sloane, A. (2005). When and How to Develop Domain-Specific Languages. *ACM Computing Surveys*, 37(4):316–344.

[Szyperski, 2002] Szyperski, C. (2002). *Component Software: Beyond Object-Oriented Programming, 2nd edition*. Addison-Wesley.