Notes on External Entities in xUML

Organizing and Understanding Bridging

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME
0.1	March 29, 2021	Initial set of notes.	GAM
0.1	October 3, 2021	Adding specific examples.	GAM

Contents

List of Figures

Introduction

This document is a set of notes about my recent thinking on bridging domains in xUML. The notes are very rough and probably create more questions than they answer. This should be considered a first rough draft for a more expansive article on this matter. Here we are concerned with basic terms and definitions. There remains much more writing and drawing to communicate the ideas effectively. Note in particular there is no model of the bridging in this document. We must get to the point where our understanding is sufficient to construct a meta-model of bridging. This document is a small step in that direction.

Basic terms

- A *system* is composed of a set of interacting *domains*.
- *Subject matter* is the basis for the decomposition of a system.
- The subject matter content of the individual domains in a system is a creative output of system design work.
- A domain may be *modeled* or *realized*. Real-world systems are almost always a mixture of both. A modeled domain is constructed by translation from an xUML domain model. A realized domain is constructed by supplying necessary interfacing to conventionally coded software.
- The decomposition of the system into domains is shown in a *domain chart*.
- The implied directed graph of a domain chart must be acyclic.
- All systems have an implied Model Execution domain which is responsible for managing the data and execution sequencing for the system, *i.e.* all systems run in the control context of a Model Execution domain.
- The decomposition of a system into domains is expected to yield a *separation of concerns*.
- The separation of concerns exhibited by the domains is expected to yield a *semantic gap* between them.
- An xUML model of a domain consists of three facets or projections of the domain subject matter.
    1. Data
    2. Dynamics
    3. Processing
- The domain is the unit of encapsulation.
- The domain is the unit of reuse.

Domain interactions

- When the domains of a system are assembled together, the details of their interaction in terms of data and control must be specified. The flow of control and data is distinct from the flow of requirements shown on a domain chart.
- From the point of view of a domain, all its outside interactions happen through one or more *external entities*.
- All domains must define at least one external entity, otherwise it has no way to interact with the outside world. Domains which produce no side effects outside their boundary serve no purpose.
- Domains may delegate aspects of their requirements to other domains. We consider such delegation as forming a client / server relationship between the participating domains.
- We consider external entities to be the model level mechanism where requirements delegation is realized.
- A domain may offer services to other domains to fulfill a client / server role between the domains.

- A domain may serve both client and server roles.
- Domain interactions fall into two broad categories:
  1. Request / response
  2. Announcements
- Request / response interactions may also be *synchronous* or *asynchronous* in nature.
- Announcement interactions are always *asynchronous*. So, an announcement is a degenerate, but worthwhile case, of an asynchronous request for which there is no response.
- A synchronous interaction blocks further execution of the invoking activity until the interaction is completed.
- An asynchronous interaction implies the invoking activity continues to execute. If a response is necessary, it happens at a some later time.
- A synchronous interaction may (or may not) return a result.
- Asynchronous interactions never returns an immediate result.
- The time nature of an interaction, *i.e.* whether it's synchronous or asynchronous, is strictly from the viewpoint of the invoking domain. The implementation is free to use any technique to implement the interaction as long as the viewpoint of the invoking domain is maintained. So, synchronous interactions may be implemented by asynchronous implementation mechanisms as long as the expected perception by the invoking domain is maintained.

External entities

- An external entity is a *named* proxy for a *domain class*.

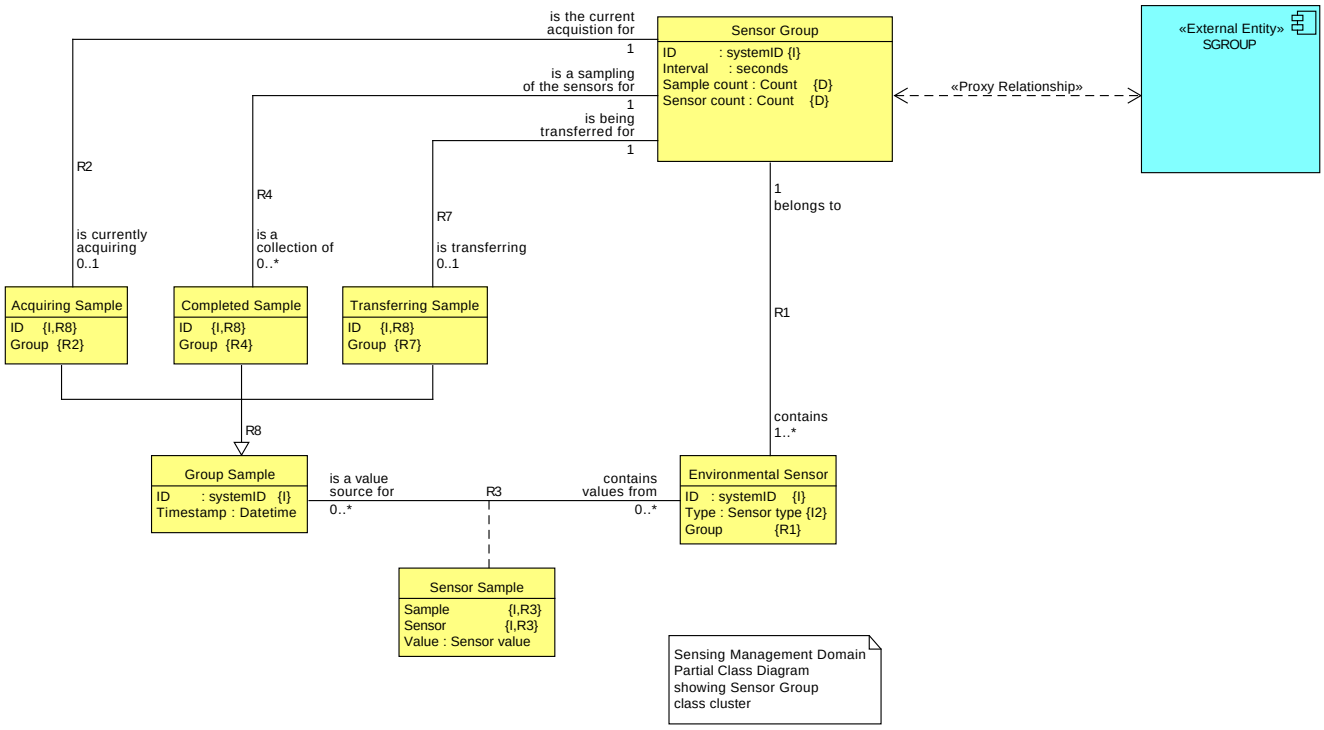

Figure 1: Sensor Group Proxy Relationship

- An external entity defines a set of *operations* it can perform.

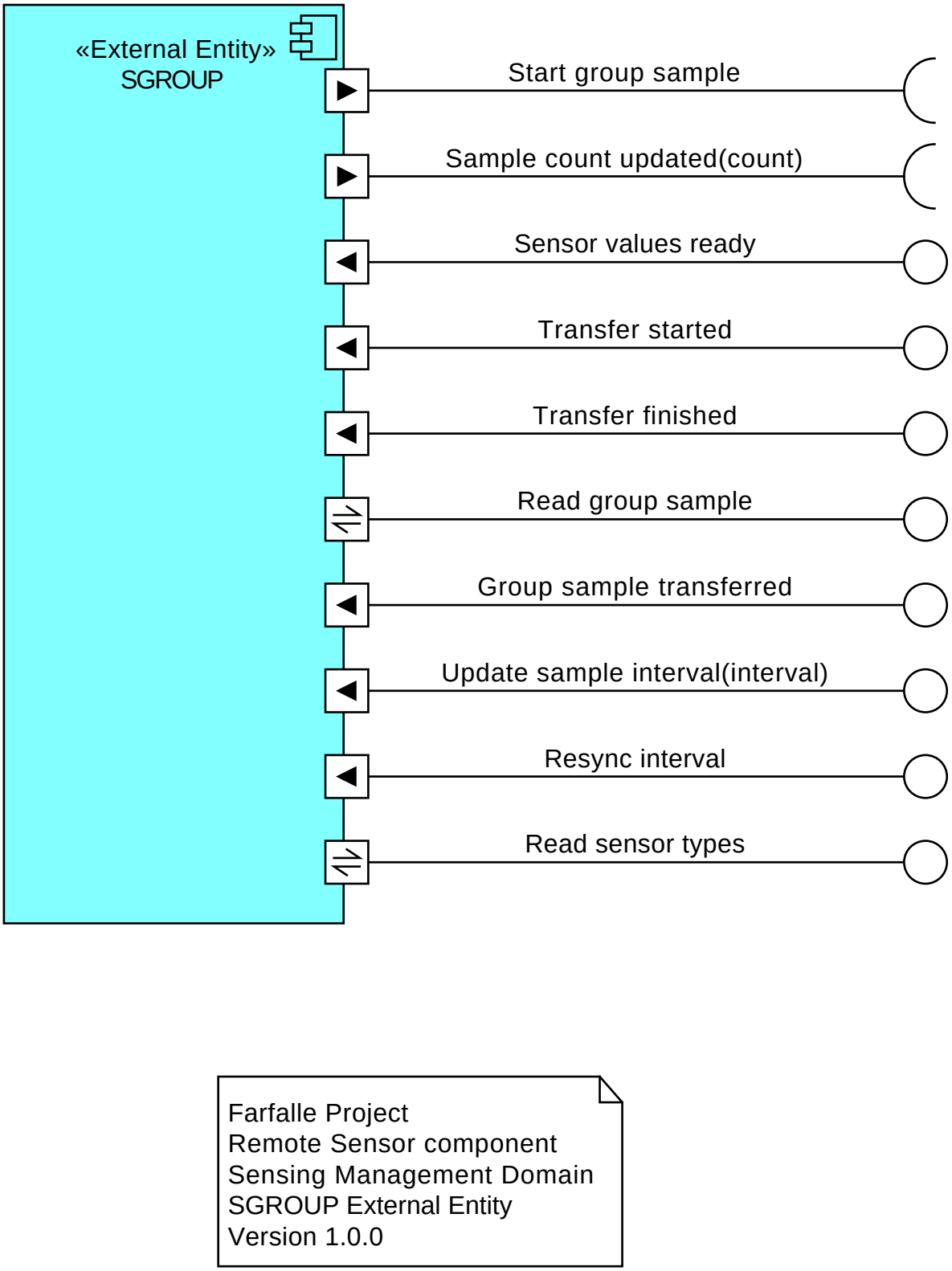

Figure 2: SGROUP Entity Operations

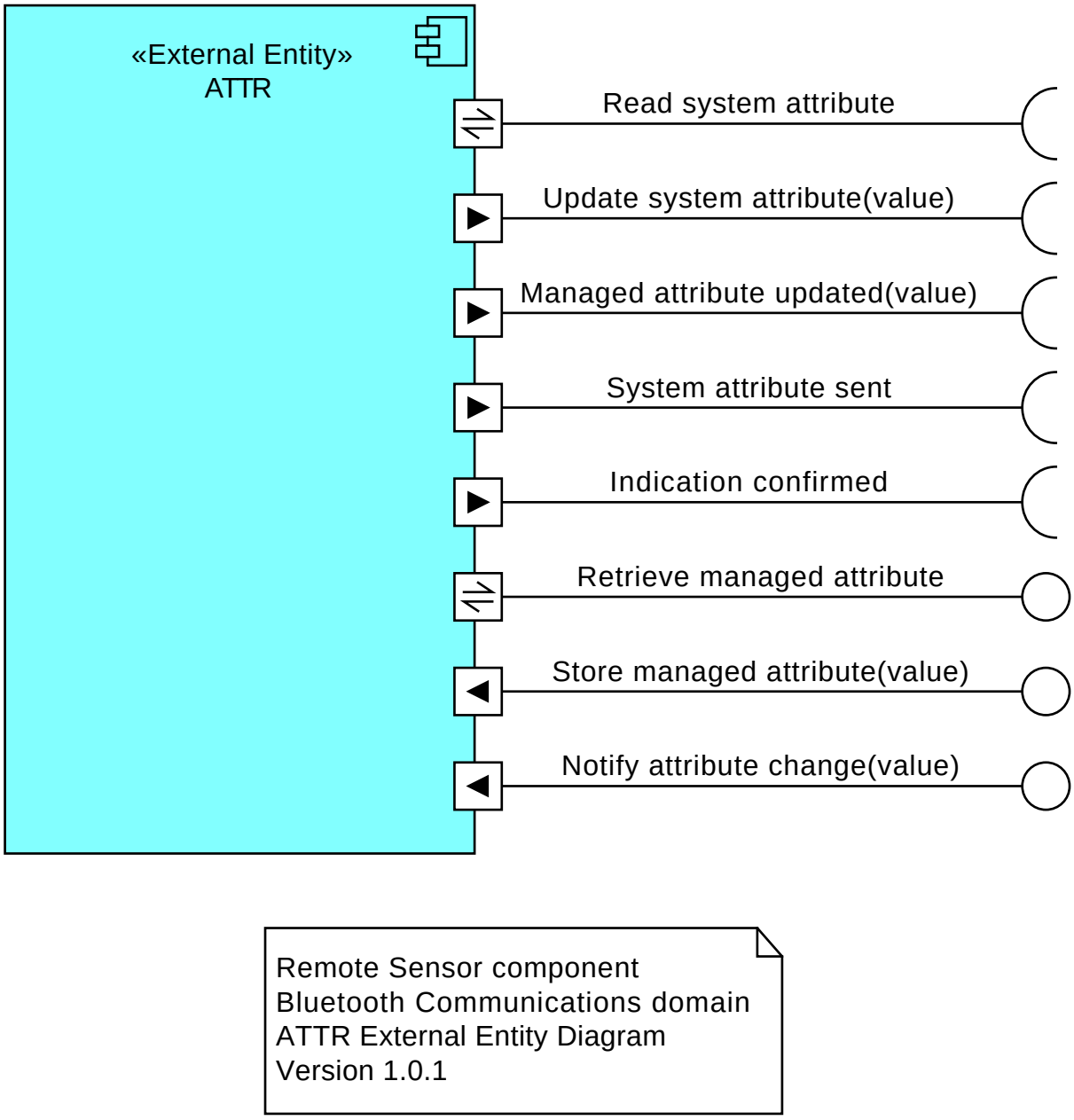

Figure 3: ATTR Entity Operations

- An external entity operation may require *parameters*.
- The semantics of an external entity operation are always expressed in the terms of the domain in which the entity is defined. Any parameter data types are those defined by the domain.
- Operations are of two types:

  1. *Egress operations* are invoked *by* the domain and transfer data and control outside of the domain.
  2. *Ingress operations* are invoked *on* the domain with data and control being transferred into the domain.

- External entity operations carry *identification* information. Frequently the identification information is the same as an identifier for the proxy class (*i.e.* the mapping between a class identifier and the external entity identification information may be the identity mapping), but, in general, there is a mapping between class identifier values and external entity identification information. For request / response operations, the mapping must be bijective. For announcement operations, the mapping need only be injective.
- The identification information for all operations of any given external entity is the same.
- Egress operations are subject to *fan out*, *i.e.* they may be wired to ingress operations of multiple other external entities.
- Ingress operations are subject to *fan in*, *i.e.* they may satisfy the wiring from multiple other external entities.
- External entity wiring is *not* allowed to be reflexive, *i.e.* the egress operation of an external entity may not be wired to an ingress operation of the same external entity.

Bridging

- A *bridge* is an association between the external entities of two domains.
- A bridge *wires* together the operations of its participant external entities.
- Wiring happens from an egress operation of one domain to an ingress operation of an other domain.
- The egress operations of an external entity are said to be *service facing*.
- The ingress operations of an external entity are said to be *client facing*.
- When external entities are wired into a bridge, the underlying classes represented by the external entities are deemed *counterpart classes*.

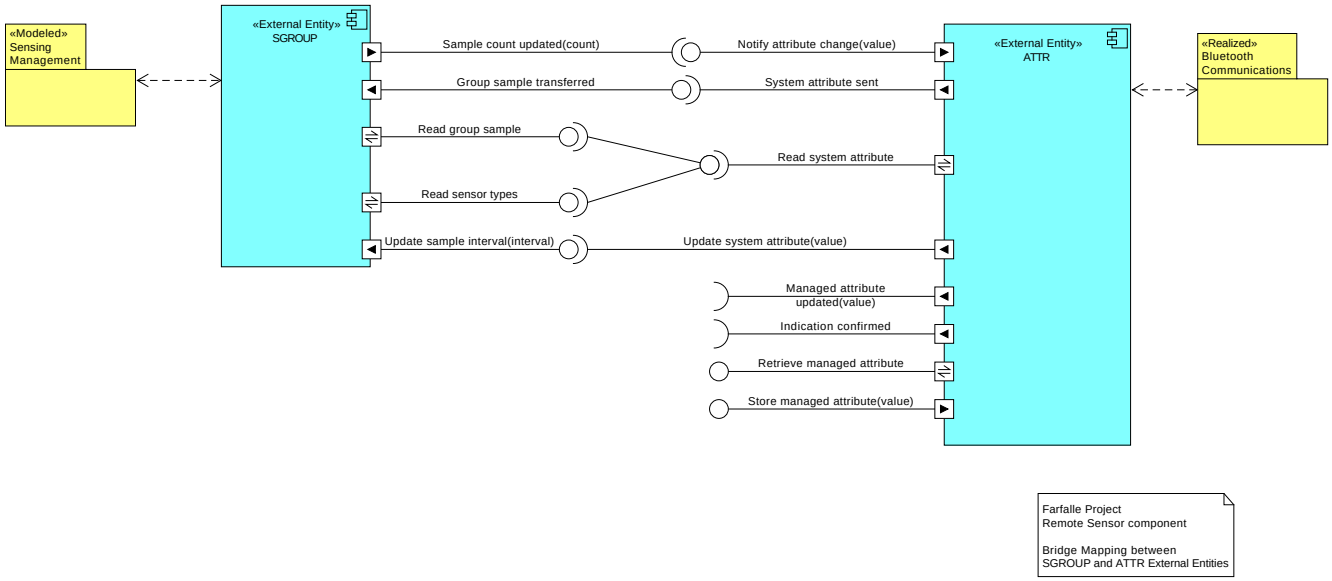

Figure 4: SGROUP / ATTR External Entity Mappings

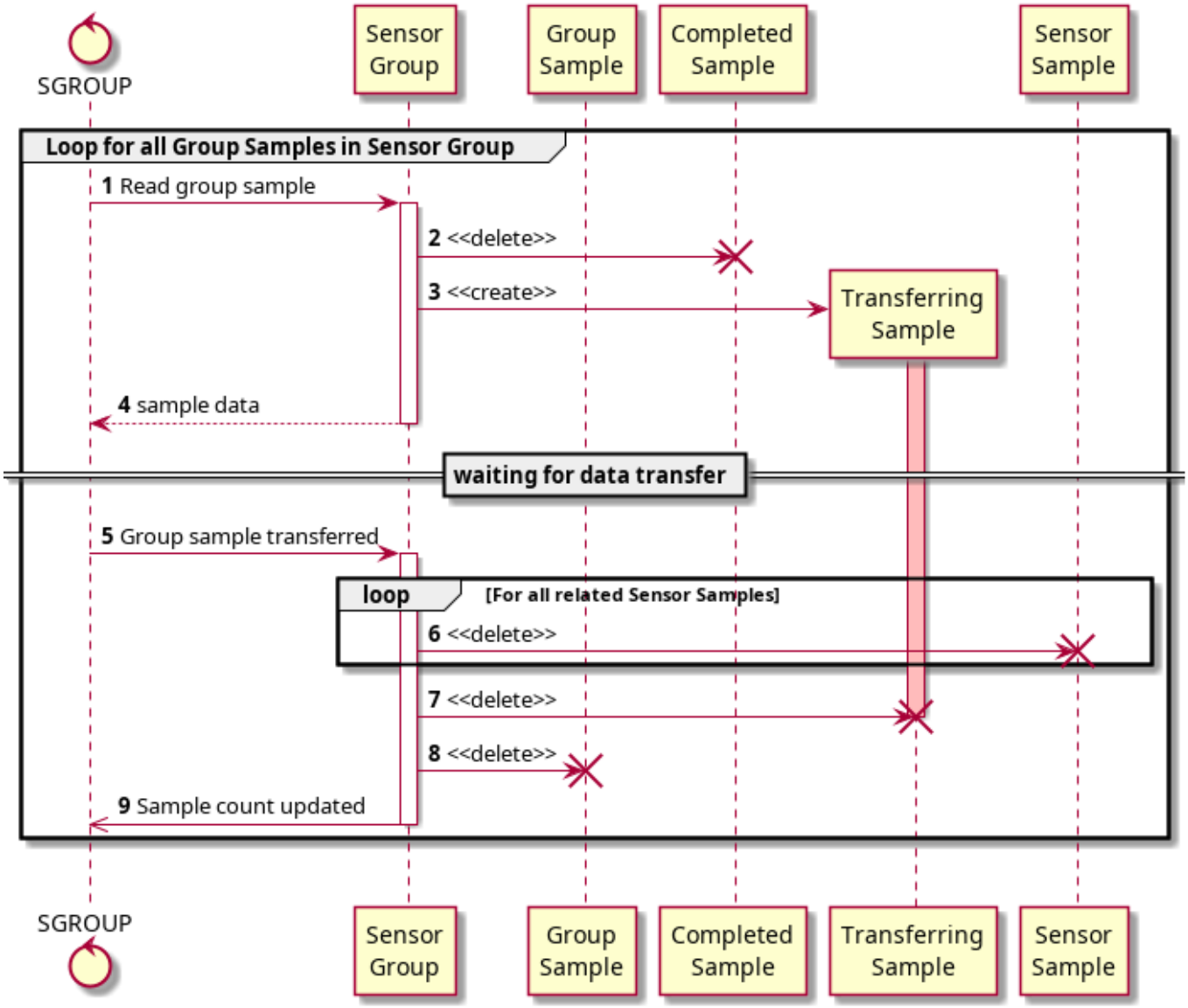

Figure 5: SGROUP Role in data transfer

External entity operation execution

- Egress operations execute outside the domain boundary.
- Ingress operations execute inside the domain boundary.
- Executing a bridge operation involves:

1. An invocation of an egress operation in an originating domain. This invocation must occur within a context where the attribute values of an identifier of the class instance are available.
2. A mapping of instance identification in the client domain to external entity identification for the egress operation.
3. A mapping of the egress entity identification to the corresponding ingress entity identification. This mapping constitutes the *semantic mapping* between the domains as it associates instances of the counterpart classes. The mapping represent a correlation table implementing a counterpart association in terms of the external entity identification information.

4. The semantic mapping is distinct for each pair of external entities, regardless of any fan in or fan out considerations. So, if a particular event announcement fans out to multiple external entities, then multiple semantic mappings are necessary.
5. The semantic mapping for a bridge may be constant at run time or it may be dynamic at run time, depending upon the life time behavior of the participating classes. For this reason, bridge construction must be done after the initial instance population of the domains is determined. If a semantic mapping is dynamic, then the necessary information to support mapping entry creation and deletion must be available from the external entity operations themselves, *i.e.* there is no *side channel* available to obtain the semantic mapping information.
6. The coercion of argument values for any external entity parameters passed out of an egress operation to compatible values in the data type for the input arguments of the ingress operation. Again, in many situation argument value mapping is the identify mapping.
7. The invocation of the ingress operation with the mapped identification information and any transformed input argument values. The ingress operation executes in the context of the receiving domain.
8. A mapping of the ingress external entity identification to an instance reference within the receiving domain.
9. The ingress operation itself is then executed in the same context as an instance based operation.

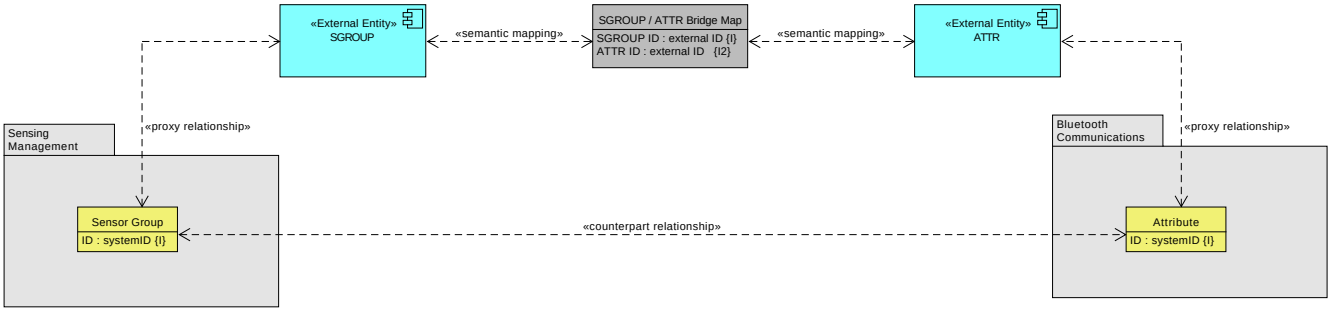

Figure 6: External Entity Semantic Mapping

Implicit bridging

- *Implicit bridging* is a specification to the Model Execution domain to invoke an external entity egress operation on the behalf of a class instance whenever a given model level operation is performed. Implicit bridging has parallels to aspect oriented program execution.
- The allowed model level operations which may be associated with an implicit bridge are:
    1. Creating a class instance.
    2. Deleting a class instance.
    3. Migrating a subclass instance.
    4. Signaling an imminent event.
    5. Signaling a delayed event.
    6. Canceling a delayed event.
    7. Transitioning to a state.
    8. Reading an attribute value (which includes computing the formula for a dependent attribute).
    9. Updating an attribute value.
    10. Invoking an instance based operation on a class instance.
- The specification of implicit bridge requests happens at system assembly time. The implicit bridging requests cannot be specified or changed at run time.

- Implicit bridging is best suited for cross-cutting system concerns and are intended to increase domain reuse by removing the handling of cross-cutting concerns from the domain activities.
- Explicit external entity invocations are best suited to situations where one domain deliberately delegates requirements to another domain and such delegation is an essential characteristic of the domain and does not significantly compromise reuse.
- This difference in usage leads to the concept of *appertained* external entities and *disjoint* external entities.
- Appertained external entities are those which are defined by a domain and must be resolved for each use of a domain in a system.
- Disjoint external entities are those which are defined only after the domain is populated and are introduced to fulfill particular cross-cutting requirements. Adding disjoint external entities to a domain along with implicitly bridging the disjoint egress operations provides a means to observe the operations of the domain without modifying the domain artifacts themselves. Although such a scheme would be subject to much abuse, it would have particular implications for better handling of cross-cutting concerns and for testing (which is the ultimate in cross-cutting concern).

Declarative bridging

- These notes view external entity operations as an executable binding between domains.
- Our preference would be to bridge domains in a declarative manner so the required implementation could be generated programmatically.
- It remains an open area as to whether specifying the bridging of domains may be accomplished reasonably using strictly declarative specifications.
- It remains to create a descriptive model of bridging to evaluate the prospects of generating bridge code from a declarative specification.