6
Case Study

This chapter shows how the strategy proposed in the previous chapter works for a realistic scenario, such as the one described in Section 2.1. We present two different examples, a client query (synchronous interaction) and a client subscription (asynchronous interaction). For both cases we assume that the context data distribution corresponds to what was shown in Figure 4.1.

6.1
Synchronous Interaction

At first, let us consider that Rule 6.1 was submitted (Step S1) by a client application as a *synchronous query*. This rule says that "if Silva is located in a room, where some activity is taking place, and if this activity has already started, then Silva is busy." The inference of this rule consists in finding values for the pair of variables "?s" and "?r", i.e., a set of tuples (s_i, r_i), that make true the atoms of the antecedent of the rule.

Rule 6.1:

$$isLocatedIn(\text{“Silva”},?r) \wedge takesPlace(?s,?r) \wedge hasStarted(?s) \longrightarrow isBusyInActivity(\text{“Silva”},?s)$$

Looking at Figure 4.1, we can notice that the context information necessary for detecting the situation described by this rule comprises data related to the user's preferences and data collected from the device (Silva's location) and information about the event (room where a session "takes place" and the fact that it "has started"). In our scenario, while the first piece of information is originated and managed at the mobile device, the latter two pieces are managed by the ambient infrastructure.

In the next step, the rule has to be partitioned (Step S2) to identify which part of R may be dealt at each side of the system. This rule would be split in the *local part* R_L, described as Rule 6.2, and the *remote part* R_R, described as Rule 6.3. Variable "?r" is an example of a variable that has to be pre-evaluated by the *local reasoner*, because the information about the predicate "isLocated"

is available only at the *user side*, but the variable is present both in R_L and R_R. In this case, the set of variables that are common in both parts of the rule consists in the unitary set $V = \{?r\}$.

Rule 6.2:

isLocatedIn("Silva",?r)

Rule 6.3:

takesPlace(?s,?r) $\wedge$ *hasStarted(?s)*

After the pre-evaluation of the variable "?r" by the *local reasoner* (Step S3), we find the set of values for "?r" as the unitary set $T = \{"Room\_B"\}$. In the next step, R_R, T and V, are forwarded to the *remote reasoner* (Step S4). The evaluation of the *forwarded rule* by the *remote reasoner* (Step S5) under the restrictions imposed by the set of values T for the variables in V is equivalent, in this case, to the evaluation of the Rule 6.4 in which "?r" assumes the value "Room_B".

Rule 6.4:

takesPlace(?s,"Room_B") $\wedge$ *hasStarted(?s)* $\longrightarrow$ *isBusyInActivity("Silva",?s)*

The result for the evaluation of this rule, considering the data available on the *ambient side* (Figure 4.1), would be the unitary set $S = \{"Session\_2"\}$. This result would be sent by the *remote reasoner* (Step S7) to the *local reasoner* (Step S6), that would send the result to the client application (Step S6), meaning that at that time, "Silva is busy participating in Session 2."

6.2 Asynchronous Interaction

Let us now consider that Rule 6.5 has been submitted by an application (Step S1) as a *subscription*. This rule says that "if an activity that Silva wants to attend is about to start and he is located in a room different from the one where the activity takes place, then he should go to that room." In this case, the inference of this rule consists in finding values for the tuples of variables "?s", "?x" and "?y" that make true the antecedent of the rule.

Rule 6.5:

wantsToAttend("Silva",?s) ∧ isLocatedIn("Silva",?y) ∧ takesPlace(?s,?x) ∧ differentFrom(?y,?x) ∧ isAboutToStart(?s) ⟶ shouldGoTo("Silva",?x)

As in the first example, the atoms in this rule comprise data available both at the *user side*, such as information about the user's preferences (sessions that Silva "wants to attend" and his location), and the *ambient side*, such as information about the activities (room where a session "takes place" and the fact that it "is about to start").

In the next step (Step S2), R would be partitioned in the *local part* R_L and the *remote part* R_R, as described by Rules 4.9 and 4.10., respectively.

Rule 6.6:

wantsToAttend("Silva",?s) ∧ isLocatedIn("Silva",?y)

Rule 6.7:

takesPlace(?s,?x) ∧ differentFrom(?y,?x) ∧ isAboutToStart(?s)

We can identify that the set of variables that are common in R_L and R_R is $V = \{?y, ?s\}$. In the pre-evaluation of R_L (Step S3) we determine that there is a single value for the variable "?y", which is "Room_B", while there are four different values that satisfy the variable "?s", which are "Session_1", "Session_2", "Session_3" and "Session_4". In practice, we have a set of four tuples that represent values for the variables in V that make R_L true, such that $T = \{($*"Room_B"*, *"Session_1"*$)$, $($*"Room_B"*, *"Session_2"*$)$, $($*"Room_B"*, *"Session_3"*$)$, $($*"Room_B"*, *"Session_4"*$)\}$.

The next step is then the forwarding of R_R (Step S4) to the *remote reasoner* with the correspondent sets V and T. In this case, the *remote reasoner* receives the *forwarded rule* as a *subscription*, and the execution of a first evaluation of the rule (Step S5) produces a void result, i.e., there is no such a tuple of values for "?y", "?s" and "?x" that make Rule 6.7 true. The *remote reasoner* adds the *forwarded rule* to a list of subscriptions and starts to monitor the context variables present in that rule (Step S8). Therefore, if there is any change in facts involving the predicates *takesPlace* or *isAboutToStart*, the rule is reevaluated (Step S9). For instance, if the fact *isAboutToStart("Session_4")* is added to the ABox on the *ambient side*, the rule R_R is reevaluated and the result would be the unitary set $S = \{$*"Room_A"*$\}$. This result would be notified

to the *local reasoner* (Step S11), which would send it to the client application, meaning that "Silva should go to Room A".

At the *local reasoner*, the *local part* of the rule R_L is put in a list of subscriptions and the respective context variables are monitored by *local reasoner* (Step S8). If any change involving the predicates *wantsToAttend* or *isLocatedIn* occurs, the rule is reevaluated (Step S9). For instance, if there is a change so that the fact *wantsToAttend("Silva","Session_2")* is removed from the ABox on the *user side*, R_L would be reevaluated and the result would be a new set T, different from the previously found (Step S10). The *local reasoner* would update this information sending the new set T with an update number to the *remote reasoner* (Step S12). As we discussed in Subsection 5.1.3, whenever a context change happens and T changes, there will be a new update associated with an exclusive update number. When a result is received from the *remote reasoner*, it carries the number that was provided with the last update, so as to enable the *local reasoner* to check if the result is valid and may be sent to the client application.

6.3 Discussion

In this chapter we exemplified how the cooperative reasoning strategy works for client queries and client subscriptions. We discussed each step of the proposed strategy, explaining the obtained results. Through this example, we could demonstrate the adequability of this approach for a common AmI scenario, as discussed in Section 5.1.3.