```
public class AccountRegistry
{
   abstract class Msg { public Guid Id { get; set; } }
   class LookupMsg : Msg { }
   class RegisterMsg : Msg
   {
      public AccountState AccountState { get; set; }
   }
}
```

Zdefiniowałem te typy komunikatów jako klasy wewnętrzne, ponieważ są one używane tylko w klasie `AccountRegistry`, aby komunikować się z jej agentem.

Możemy teraz zdefiniować metodę `Lookup`, która definiuje publiczne API dla `AccountRegistry` i dlatego jest wykonywana na wątku wywołującego, jak poniżej:

```
public class AccountRegistry
{
   Agent<Msg, Option<Account>> agent;
   Func<Guid, Task<Option<AccountState>>> loadState;

   public Task<Option<Account>> Lookup(Guid id)
      => agent
         .Tell(new LookupMsg { Id = id })
         .OrElse(() =>
            from state in loadState(id)
            from result in agent.Tell(new RegisterMsg
            {
               Id = id,
               AccountState = state
            })
            select result);
   }
}
```

Mówi agentowi, aby odszukał dany ID

Jeśli wyszukiwanie się nie powiedzie, stan jest ładowany w wątku wywołującego

Mówi agentowi, aby zarejestrował nowy proces z danym stanem i ID

Listing 15.9 Potrzeba różnych typów komunikatów, aby przekazać intencję wywołującego

Najpierw prosi agenta o odszukanie ID; jeśli wyszukiwanie się nie powiedzie, wtedy stan jest pobierany z BD. Zwróćmy uwagę, że jest to zrobione na wątku wywołującego i zwalnia agenta od obsługi innych komunikatów. Na końcu drugi komunikat jest wysyłany do agenta z prośbą o utworzenie i zarejestrowanie `AccountProcess` z danym stanem konta i ID.

Zwróćmy uwagę, że wszystko dzieje się na złożeniu `Task<Option<>>`, ponieważ jest to typ zwracamy zarówno przez `loadState`, jak i przez `Tell`. Nawet `OrElse` wykonuje przeciążenie zdefiniowane w `Task<Option<T>>`, które wykonuje podaną funkcję awaryjną, jeśli `Task` nie zostanie wykonane *lub* jeśli wewnętrzna wartość `Option` to `None`.

Pozostała tylko do pokazania definicja agenta w konstruktorze `AccountRegistry`.