If we map the above definitions onto use case specifications we can observe that they can fully encompass the system's application logic. This involves use case models with actors, relationships between use cases, and use case scenarios. However, in order for the application logic to be defined unambiguously, scenario sentences have to refer precisely to domain notions and user interface elements (see e.g. previous work on use cases and domain vocabularies[32]). Considering all the above, the use case language (in further descriptions we will use the acronym UCL) would have the following syntactical constructs:

- actors - roles of entities (humans, machines or events) outside of the specified system that interact with the system;
- use cases - logical units of system's application logic, leading to specific goals of value to some actor;
- use case invocations - relationships between use cases that determine possible invocation of logic contained in one use case by another use case;
- scenarios - sequences of sentences that determine possible flows of actions that start with an initial actor interaction and end when a use case goal is achieved or fails;
- action sentences - sentences that control the state of dialogue between the system and its actor(s), and passing of data between them; they refer to either domain logic actions (data processing) or user interface actions;
- control flow sentences - sentences that control flow of execution of actions (including invocation of other use cases) during use case execution (instantiation);
- actions - phrases that define units of domain logic (data processing) or user interface behaviour; they always refer to a notion being part of the problem domain or of the user interface;

To present the UCL syntax, we will use an example use case model with associated scenarios. For the sake of brevity we will omit formal definition of the syntax which is quite simple and is an extension and improvement of that found in our previous work [33]. We will first discuss the syntax and associated action and control flow semantics in an informal way. In the following sections we will present formal translational semantics through defining a translational framework and giving respective translation rules.

The example is quite small but shows most of the syntactical elements. Figure 1 presents the graphical part of the syntax which is close to the UML syntax for use case models (see the UML specification [26], Chapter 18). As we can notice, use case ovals can be related through special «invoke» relationships. These relationships substitute fuzzy «include» and «extend» relationships (see e.g. work by van den Berg and Simons [36] and Laguna et al. [21, 22]) found in the standard UML. Invocations in UCL have call semantics, where the invoking use case can "call" the invoked use case at some point (or points) within its scenarios. The invoked use case can return a status value that might influence the actual flow of interactions within the invoking use case.

Obviously, the models involve actors that can be related to use cases. There are two types of such relationships: usage and participation. The first one is denoted by an arrow pointing from an actor to a use case (e.g. "Cashier" –> "Find client" in Figure 1). In this case, an actor "uses" a use case and constitutes the main actor that starts the use case. At the same time, the use case becomes available to the actor, forming part of the actor's access permissions. Usually, such use cases are available to the related actors through some options (e.g in the menus) in the user interface. The second – participation – relationship is expressed by an arrow pointing from a use case to an actor (e.g. "Acknowledge money transfer" –> "Manager" in Figure 1). It denotes that some actor takes part in interactions defined in the related use case but does not initiate it. This usually means, that the actor is somehow prompted to perform some actions and/or input some data. This prompting is

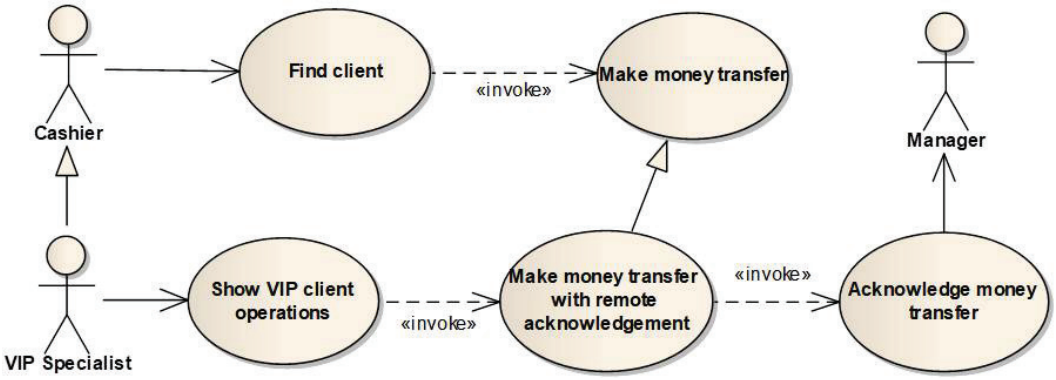

Fig. 1. Example use case model

not caused by the "participating" actor but is a consequence of previous interactions of the system with other actors.

Use case models can also include generalisation/specialisation relationships – both between use cases and actors. Specialisation between use cases means that some use case adds to the functionality of some general use case. Specifically, specialised use cases define additional scenarios that allow for additional alternative flows of interactions. This means that a specialised use case extends the functionality of a general use case. A special situation, illustrated in Figure 1, is when a general use case (here: "Make money transfer"), specialised by other use cases (here: "Make money transfer with remote acknowledgement") is invoked. In this case, invoking a general use case means potential invocation of one of the specialised use cases. The decision on which use case is to be invoked depends on respective use case preconditions and will be explained when discussing scenario syntax and semantics below.

Specialisation between actors denotes inheritance of access rights. Specifically, this means that a specialised actor has access to all the use cases in usage and participation relationships of the general actor. In addition, the specialised actor has – obviously – access rights to use cases related to that actor. For example, the actor "Cashier" in Figure 1 will have the option to start the "Find client" use case in his/her main menu. The same option will be available to the "VIP Specialist" actor, and in addition he/she will have the option to start "Show VIP client operations".

The key part of the UCL syntax is the syntax for scenarios. The main idea was to retain comprehensibility of the syntax through a synthesis of existing use case content templates. Most of such templates use natural language sentences written with varying degree of formality. We follow this path by retaining such elements of typical use case notations as preconditions, postconditions, scenarios and alternative flows. At the same time, we constrain the language to assure its precision and facilitate translation into fully usable and operational code.

Let us now analyse the definition of scenarios for the "Find client" use case in our running example. The definition starts with the use case name and at then defines the main scenario. This scenario starts with a precondition placed in curly brackets. This is followed by a sequence of labelled sentences that determine the flow of interactions. This sequence is ended with a statement that defines the final status value returned on termination of the use case flow. Alternative scenarios begin following the main scenario. Each such scenario starts by repeating one of the sentences in another scenario. Then, it is followed by a sequence of sentences and is ended with a final statement.

```
Use case Find client Main scenario
{cashier logged-in}
00: Cashier selects find client <<trigger>>
01: System <<shows>> client search form <<screen>>
02: Cashier enters client search <<data>>
03: Cashier selects search <<trigger>>
04: System <<closes>> client search form <<screen>>
05: System retrieves client <<data>>
06: System <<shows>> client window <<screen>>
07: Cashier selects close <<trigger>>
08: System <<closes>> client window <<screen>>
-> [ended]

Scenario
05: -"-
A1: Cashier <<invokes>> make money transfer <<use case>>
[success]
-> rejoin 05

Scenario
A1: -"-
[not success]
-> rejoin 06
```

To analyse this syntax in more detail we first define sentence types, and present their syntax.

- **Action sentences**. These sentences follow the Subject-Verb-Object syntax. The subject is either a specific actor name (e.g. "Cashier") or one of the keywords: "Actor" or "System". The rest of the sentence is a reference to an action that is composed of a verb and a noun. The noun refers either to a domain notion («data»), or a user interface element («trigger», «screen») or a use case name («use case»). The verb denotes any operation related to the noun. Some verbs use standard pre-defined keywords («shows», «closes», «invokes»). In all cases, each sentence part can be a phrase composed of several words (e.g. "client window"). Action sentences are labelled with a label (usually a consecutive number) followed by a colon.
- **Precondition sentences**. Such sentences can be placed only at the beginning of the main scenario for a given use case. They contain a set of conditions placed within curly brackets. Each condition is composed of an actor or a noun (just like a Subject or a Verb in an action sentence), and an adjective. The adjective refers to some query associated with the given actor or noun.
- **Alternative flow sentences**. Such sentences always begin alternative scenarios. They contain a label of a sentence in another scenario followed by the ditto symbol "-"-".
- **Condition sentences**. Such sentences contain some text in square brackets. This text is not restricted but should match status values returned by actions from the preceding action sentence. It is also allowed to precede this status text with a keyword "**not**".
- **Final sentences**. Such sentences can be placed only at the end of scenarios. Just like condition sentences, they contain free text that defines a status value that is returned as the given use case reaches the particular final sentence.
- **Rejoin sentences**. Also rejoin sentences can be placed at the end of scenarios. They indicate an action sentence (its label) in another scenario of the same use case to which flow of control is moved after reaching the particular rejoin sentence.

In addition to following this syntax, scenarios have to follow well-formedness rules. These rules determine sequences of scenario sentences that are valid. To define the rules, we introduce the notion of **state of dialogue**. There are two possible states: "actor" and "system". Additionally, the "actor" state can be supplemented with the actor name. At the beginning of the main scenario, the state of dialogue is "actor" and the actor name is the name of the main actor for this use case (if available). Every action sentence can be used only in a specific state of dialogue. Moreover, action sentences can change this state according to their type. The following list defines possible types of action sentences and their well-formedness rules.

- **Actor-to-trigger**. The subject of the sentence is an actor, and the object is a «trigger» noun. Can be used in the state of dialogue "actor", and changes the state of dialogue to "system".
- **Actor-to-data**. The subject of the sentence is an actor, and the object is a «data» noun. Can be used in the state of dialogue "actor", and does not change the state of dialogue.
- **Actor-to-use-case**. The subject of the sentence is an actor, and the object is a «use case» noun. Can be used in the state of dialogue "actor", and does not change the state of dialogue.
- **System-to-screen**. The subject of the sentence is the system, and the object is a «screen» noun. Can be used in the state of dialogue "system", and changes the state of dialogue to "actor".
- **System-to-message**. The subject of the sentence is the system, and the object is a «message» noun. Can be used in the state of dialogue "system", and does not change the state of dialogue.
- **System-to-data**. The subject of the sentence is the system, and the object is a «data» noun. Can be used in the state of dialogue "system", and does not change the state of dialogue.
- **System-to-use-case**. The subject of the sentence is the system, and the object is a «use case» noun. Can be used in the state of dialogue "system", and does not change the state of dialogue.

In addition to this, the following rules apply.

- Sentences other than action sentences do not change the state of dialogue.
- The first sentence in the main scenario has to be an actor-to-trigger sentence.
- The last sentence in every scenario has to be either a final sentence or a rejoin sentence.
- The rejoin sentence has to point to a sentence that is valid regarding the state of dialogue.
- Condition sentences can only follow system-to-data, system-to-use-case or actor-to-use-case sentences.
- Every alternative scenario has to begin with an alternative flow sentence, and such sentences can only be used for this purpose.
- Every main scenario can (but may not) begin with a precondition sentence.

Considering these rules for forming use case scenarios we can now analyse the example scenarios of the "Find client" use case above. Notice that the notation tries to follow natural language as closely as possible. The use case starts with an initial triggering event that often reflects pressing a menu option or button (here: selecting the "make transfer" option). THe following sequence

```
Use case Make money transfer Main scenario
{client <<data>> selected; actor authorised for transfer}
00: Actor selects make transfer <<trigger>>
01: System retrieves client accounts <<data>>
02: System <<shows>> money transfer form <<screen>>
03: Actor enters money transfer <<data>>
04: Actor selects transfer <<trigger>>
05: System validates money transfer <<data>>
[money transfer valid]
```

```
06: System performs money transfer <<data>>
07: System shows transfer success <<message>>
08: System <<closes>> money transfer form <<screen>>
-> [success]

Scenario
03: -"-
B1: Actor selects cancel <<trigger>>
-> [cancelled]

Scenario
05: -"-
[money transfer invalid]
C1: System retrieves money transfer errors <<data>>
-> rejoin 02

Scenario
05: -"-
[money transfer above limit]
D1: System <<shows>> manager intervention window <<screen>>
D2: Actor selects abort <<trigger>>
D3: System <<closes>> manager intervention window <<screen>>
-> [aborted]

Scenario
05: -"-
[money transfer above limit]
E1: System <<activates>> card reader <<device>>
E2: Actor swipes card <<data>>
E3: System validates card <<data>>
[card valid]
E4: System <<deactivates>> card reader <<device>>
E5: System <<closes>> manager intervention window <<screen>>
-> rejoin 06

Scenario
E3: -"-
[card invalid]
F1: System retrieves card errors <<data>>
-> rejoin D1

Scenario
E3: -"-
[card invalid]
-> rejoin E2

Use case Make money transfer with remote acknowledgement
specializes Make money transfer
{client <<data>> remote acknowledgement is on} Scenario
05: -"-
[money transfer above limit]
```

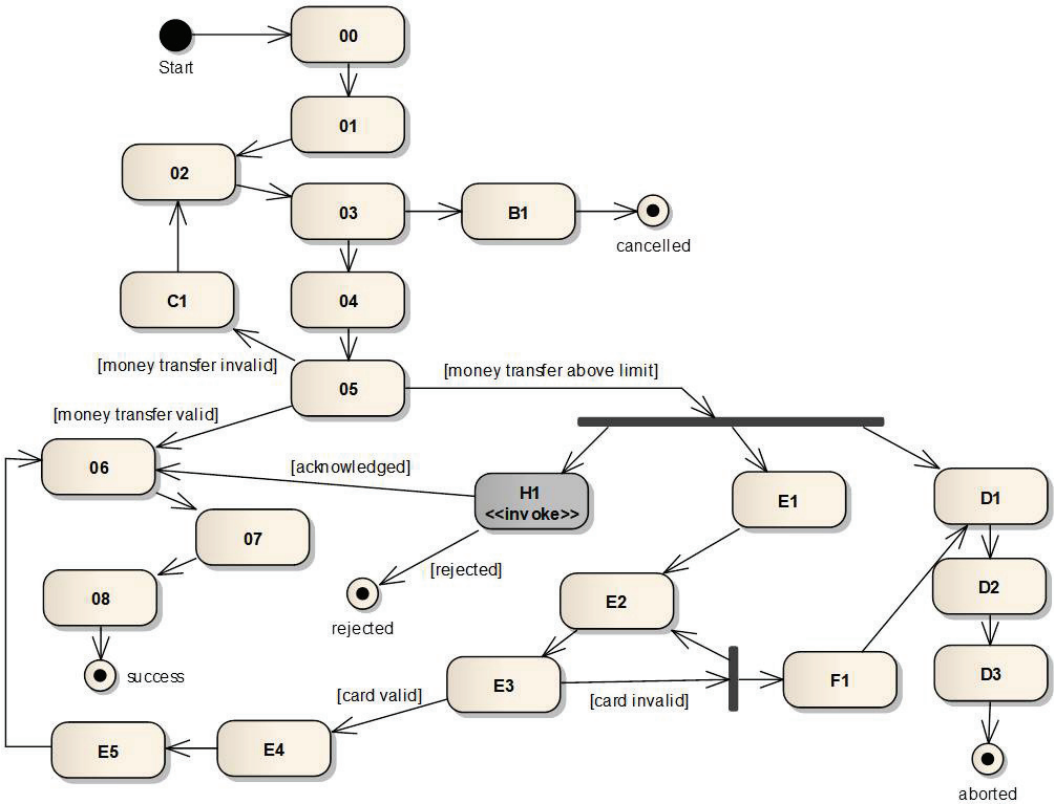

Fig. 2. Ativity diagram for the “Make money transfer” use case and its specialisation

```
H1: System <<invokes>> Acknowledge money transfer <<use case>>
[acknowledged]
-> rejoin 06

Scenario
I1: -"-
[rejected]
-> [rejected]

Use Case Acknowledge money transfer Main scenario
{any manager logged in; money transfer <<data>> selected}
00: Actor selects acknowledge transfer <<trigger>>
01: System picks manager <<actor>>
02: System <<shows>> acknowledge transfer window <<screen>>
03: Manager selects acknowledge <<trigger>>
04: System <<closes>> acknowledge transfer window <<screen>>
-> [acknowledged]

Scenario
03: -"-
```