McGill

Action Semantics for an Executable UML

Thomas Feng

March 2, 2003

Email: thomas@email.com.cn
Homepage: http://moncs.cs.mcgill.ca/people/tfeng/

MSDL

- **Why are we interested in semantics?**

Other than syntax, the pure appearance of a language, we are also interested in semantics.

▶ Why are we interested in semantics?

Other than syntax, the pure appearance of a language, we are also interested in semantics.

- A rigorously defined semantics makes our programs/models software-platform-independent [MTAL98]. Without such a semantics, it is impossible for a program/model to be easily ported to another environment.

▶ Why are we interested in semantics?

Other than syntax, the pure appearance of a language, we are also interested in semantics.

- A rigorously defined semantics makes our programs/models software-platform-independent [MTAL98]. Without such a semantics, it is impossible for a program/model to be easily ported to another environment.
- With a rigorous semantics, compiler of a specific programming language can be automatically generated.

▶ Why are we interested in semantics?

Other than syntax, the pure appearance of a language, we are also interested in semantics.

- A rigorously defined semantics makes our programs/models software-platform-independent [MTAL98]. Without such a semantics, it is impossible for a program/model to be easily ported to another environment.
- With a rigorous semantics, compiler of a specific programming language can be automatically generated.
- With a rigorous semantics, models can be tested in its design phrase. Automatic tools allow designers to prove their properties, analyze them and finally generate code.

▶ What is action semantics?

- A framework for the formal description of programming languages.

▶ What is action semantics?

- A framework for the formal description of programming languages.
- A hybrid of denotational and operational semantics. [Mos02]

▶ What is action semantics?

- A framework for the formal description of programming languages.
- A hybrid of denotational and operational semantics. [Mos02]
- Initially developed by Peter D. Mosses at University of Aarhus, early 1990's.

▶ What is action semantics?

- A framework for the formal description of programming languages.
- A hybrid of denotational and operational semantics. [Mos02]
- Initially developed by Peter D. Mosses at University of Aarhus, early 1990's.
- Goal: to give complete formal descriptions of programming languages and to use these for generating various tools, such as parsers, static analyzers, interpreters, and compilers.

▶ Advantage (1)

- Compared with denotational semantics:

▶ Advantage (1)

- Compared with denotational semantics:
  1. Denotational semantics, though very precise and formal, obscures semantics structures.

▶ Advantage (1)

- Compared with denotational semantics:
  1. Denotational semantics, though very precise and formal, obscures semantics structures.
  2. Action semantics is modular and thus reusable.

▶ Advantage (1)

- Compared with denotational semantics:

  1. Denotational semantics, though very precise and formal, obscures semantics structures.
  2. Action semantics is modular and thus reusable.
  3. Action semantics is extensible.

▶ Advantage (1)

- Compared with denotational semantics:
    1. Denotational semantics, though very precise and formal, obscures semantics structures.
    2. Action semantics is modular and thus reusable.
    3. Action semantics is extensible.
    4. Using a language quite like natural English language, action-semantic descriptions (ASDs) are much more readable than denotational descriptions.

▶ Advantage (1)

- Compared with denotational semantics:
  1. Denotational semantics, though very precise and formal, obscures semantics structures.
  2. Action semantics is modular and thus reusable.
  3. Action semantics is extensible.
  4. Using a language quite like natural English language, action-semantic descriptions (ASDs) are much more readable than denotational descriptions.
- ASDs scale up smoothly to realistic programming languages:
  1. Provides data types (like **byte, int, float, char**) and a mechanism to define and manipulate customized data types.

▶ Advantage (1)

- Compared with denotational semantics:

  1. Denotational semantics, though very precise and formal, obscures semantics structures.
  2. Action semantics is modular and thus reusable.
  3. Action semantics is extensible.
  4. Using a language quite like natural English language, action-semantic descriptions (ASDs) are much more readable than denotational descriptions.

- ASDs scale up smoothly to realistic programming languages:

  1. Provides data types (like **byte, int, float, char**) and a mechanism to define and manipulate customized data types.
  2. Provides primitive actions to describe primitive semantic structures.

- **Advantage (2)**

  Example: defining the execution sequence of two actions.

▶ Advantage (2)

Example: defining the execution sequence of two actions.

- Action semantics:

A1 then A2

▶ Advantage (2)

Example: defining the execution sequence of two actions.

- Action semantics:

A1 then A2

- Denotational semantics (λ-notation):

$$\lambda \varepsilon_1 . \lambda \rho . \lambda \kappa . A_1 \varepsilon_1 \rho (\lambda \varepsilon_2 . A_2 \varepsilon_2 \rho \kappa)$$

Denotational semantics, though formal and rigorous, is usually much more complex than action semantics. [Mos96]

- **Action semantics for UML**

UML only defines the syntax of models. The semantics, though informally described in a plain natural language, is not precise enough to specify model behavior. Thus different (meta-)modelling tools have their own interpretation, limiting the portability and reusability of models.

▶ Action semantics for UML

UML only defines the syntax of models. The semantics, though informally described in a plain natural language, is not precise enough to specify model behavior. Thus different (meta-)modelling tools have their own interpretation, limiting the portability and reusability of models.

Action semantics, as a new semantics originally aimed at describing programming languages and automatic generation and analysis of compilers, was proposed to the OMG as an additional package for UML. [AILKC$^+$00]

▶ Action semantics for UML

UML only defines the syntax of models. The semantics, though informally described in a plain natural language, is not precise enough to specify model behavior. Thus different (meta-)modelling tools have their own interpretation, limiting the portability and reusability of models.

Action semantics, as a new semantics originally aimed at describing programming languages and automatic generation and analysis of compilers, was proposed to the OMG as an additional package for UML. [AILKC+00]

It is nicely compatible with other components in UML, including the Object Constraint Language (OCL). With the OCL extension, ASDs are allowed to use the OCL syntax, i.e. to navigate among the objects with the OCL dot-notation.

▶ Control flow and data flow (1)

Two kinds of flows control the execution sequence of actions: control flow and data flow.

▶ Control flow and data flow (1)

Two kinds of flows control the execution sequence of actions: control flow and data flow.

- Control flow. An action is executed only after all its antecedents are completed.

▶ Control flow and data flow (1)

Two kinds of flows control the execution sequence of actions: control flow and data flow.

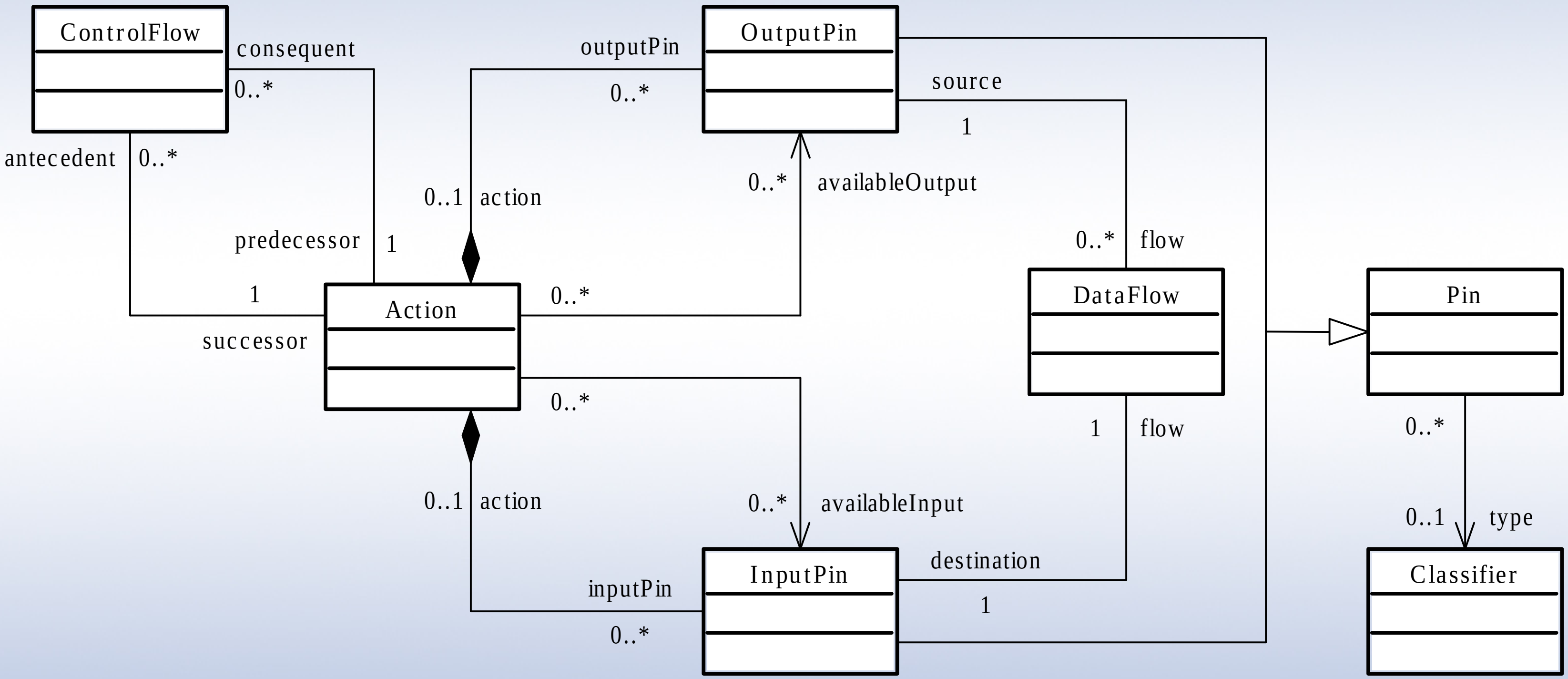
- Control flow. An action is executed only after all its antecedents are completed.
- Data flow. Every action has two sets of input pins (A and A') and two sets of output pins (B and B') associating with it. One of the input pin sets contains all the required input pins for the action. The other input pin set contains available pin data at a certain time. Only when $A = A'$ can the action be executed. Similarly, only when the action is completed does B equal to B'.

▶ Control flow and data flow (2)

As the data flow carries on, data from the output pins of a preceding action become a source of the data flow, and then the confluence reaches the input pins of another action.

All actions are treated as executing concurrently unless explicitly sequenced by a flow of data or control [AILKC$^+$00].

▶ Control flow and data flow (3)



MSDL

▶ Primitive actions

For each kind of actions (which we will see later), there is a set of primitive actions. They are the atom of actions.

Primitive actions are defined in the level of action semantics. They are used to make up more complex actions.

As we have seen, **then** is a primitive action meaning first execute action **A1** and then execute **A2**.

▶ Composite actions (1)

Composite actions allow the composition of simpler actions into more complex ones. They are recursive structures.

▶ Composite actions (1)

Composite actions allow the composition of simpler actions into more complex ones. They are recursive structures.

- Group actions. Actions are allowed to group together to represent a specific design concern. Data flow must be directly connected to the input pins of the actions in the group, because group actions have not input pins. Control flow may also cross the group boundary and be directly connected to subactions.

▶ Composite actions (1)

Composite actions allow the composition of simpler actions into more complex ones. They are recursive structures.

- Group actions. Actions are allowed to group together to represent a specific design concern. Data flow must be directly connected to the input pins of the actions in the group, because group actions have not input pins. Control flow may also cross the group boundary and be directly connected to subactions.

  When group actions are physically connected with control flow, they are placed in an execution sequence with their predecessors and successors.

▶ Composite actions (2)

- Conditional actions are compositions of clauses, each of which contains exactly one test action and one body action.

▶ Composite actions (2)

- Conditional actions are compositions of clauses, each of which contains exactly one test action and one body action.

  A test action accepts input data from the available input pins, and produce a truth value. Its associated body action is executed if and only if the test result is $true$.

▶ Composite actions (2)

- Conditional actions are compositions of clauses, each of which contains exactly one test action and one body action.

  A test action accepts input data from the available input pins, and produce a truth value. Its associated body action is executed if and only if the test result is *true*.

  For each execution of a conditional action there must be exactly one test action returning the *true* value, while all the other test actions return *false*.

▶ Composite actions (2)

- Conditional actions are compositions of clauses, each of which contains exactly one test action and one body action.

  A test action accepts input data from the available input pins, and produce a truth value. Its associated body action is executed if and only if the test result is $true$.

  For each execution of a conditional action there must be exactly one test action returning the $true$ value, while all the other test actions return $false$.

  Clauses can have noncyclic predecessor-successor relationship among them. If any of its predecessors is executed, the successor clause could not be executed. Test actions of unrelated clauses may be executed concurrently.

▶ Composite actions (3)

- Loop action. The loop action provides for repeated execution of a contained action so long as a test action results in an appropriate value. It contains exactly one clause of a test action and a body.

- **Read and write actions**

Read actions retrieve values from objects, attributes, links and variables, while write actions write values to them.

▶ Read and write actions

Read actions retrieve values from objects, attributes, links and variables, while write actions write values to them.

- Object actions. Read/write the classifier of an object (given from an input pin), and produce an output only if it is a read action.

▶ Read and write actions

Read actions retrieve values from objects, attributes, links and variables, while write actions write values to them.

- Object actions. Read/write the classifier of an object (given from an input pin), and produce an output only if it is a read action.
- Attribute actions. Read/write an attribute of an object (given from an input pin). The action is statically associated with a certain attribute.

▶ Read and write actions

Read actions retrieve values from objects, attributes, links and variables, while write actions write values to them.

- Object actions. Read/write the classifier of an object (given from an input pin), and produce an output only if it is a read action.
- Attribute actions. Read/write an attribute of an object (given from an input pin). The action is statically associated with a certain attribute.
- Association actions.
  Read: accept $n-1$ objects and output the nth object of a link.
  Write: perform limited actions (destroy/reorder) on a link.

▶ Read and write actions

Read actions retrieve values from objects, attributes, links and variables, while write actions write values to them.

- Object actions. Read/write the classifier of an object (given from an input pin), and produce an output only if it is a read action.

- Attribute actions. Read/write an attribute of an object (given from an input pin). The action is statically associated with a certain attribute.

- Association actions.
  Read: accept $n-1$ objects and output the nth object of a link.
  Write: perform limited actions (destroy/reorder) on a link.

- Variable actions. Statically associated with variables.

- **Computation actions**

  Computation actions take in input values, perform pure functional computation on them, and then return output values to the output pins. They are comparable to the *constant* functions in programming languages, which are self-contained and make no change to the current state.

▶ Computation actions

Computation actions take in input values, perform pure functional computation on them, and then return output values to the output pins. They are comparable to the *constant* functions in programming languages, which are self-contained and make no change to the current state.

Action semantics leaves computation actions as an implementation-dependent part. Namely, they can be written in any specific programming language, as long as the modelling tool permits.

▶ Collection actions (1)

An attribute of an object can be a collection, ordered or unordered. A collection action applies its subaction on all the elements in the collection at a time.

▶ Collection actions (1)

An attribute of an object can be a collection, ordered or unordered. A collection action applies its subaction on all the elements in the collection at a time.

- Filter actions. A filter action applies its subaction — a test action — on each element in an *unordered* collection. Only those resulting in $true$ is returned.

▶ Collection actions (1)

An attribute of an object can be a collection, ordered or unordered. A collection action applies its subaction on all the elements in the collection at a time.

- Filter actions. A filter action applies its subaction — a test action — on each element in an *unordered* collection. Only those resulting in $true$ is returned.

- Iterate actions. The subaction is performed on each element in the *ordered* collection in sequence.

▶ Collection actions (1)

An attribute of an object can be a collection, ordered or unordered. A collection action applies its subaction on all the elements in the collection at a time.

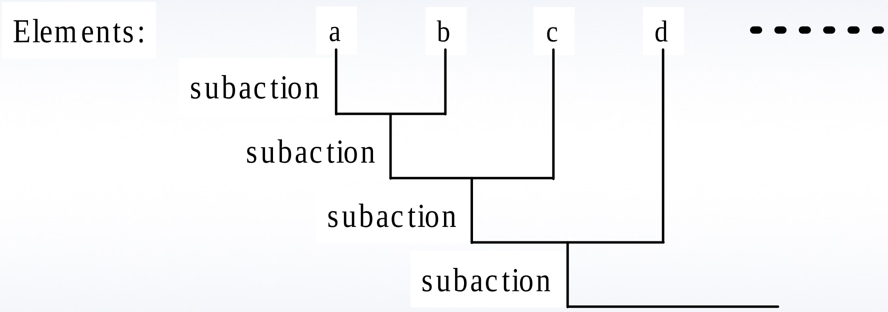
- Filter actions. A filter action applies its subaction — a test action — on each element in an *unordered* collection. Only those resulting in $true$ is returned.

- Iterate actions. The subaction is performed on each element in the *ordered* collection in sequence.

- Map actions. Elements in the *unordered* collection is functionally mapped to the results of the subaction.

▶ Collection actions (2)

- Reduce actions. The subaction is executed for sequentially for pairs of input elements, until a single output is produced.



- **Message actions**

Messaging actions provide a mechanism to facilitate invocation between objects.

▶ Message actions

Messaging actions provide a mechanism to facilitate invocation between objects.

At model time, messages are modelled as classes, with attributes as parameters or return values. Each message class designates a specific request from the requestor.

▶ Message actions

Messaging actions provide a mechanism to facilitate invocation between objects.

At model time, messages are modelled as classes, with attributes as parameters or return values. Each message class designates a specific request from the requestor.

Parameters and return values of a request or a reply are stored in an instance of the message class. Sometimes a request does not need any parameter or an invocation does not return any value (asynchronous requests described below are an example), but the message instance must be sent or received.

- **Asynchronous messages**

When an object executes an asynchronous action with a message as parameter, it starts an asynchronous invocation. The action returns no value and the requesting object does not wait for a reply from the requested object. When the requesting object and the requested object are on different machines, the requesting one even does not wait until the other object actually receives the message.

- **Asynchronous messages**

When an object executes an asynchronous action with a message as parameter, it starts an asynchronous invocation. The action returns no value and the requesting object does not wait for a reply from the requested object. When the requesting object and the requested object are on different machines, the requesting one even does not wait until the other object actually receives the message.

On receiving the message, the host object might queue it according to a specific strategy, which is not in the scope of the basic action semantics.

▶ Asynchronous messages

When an object executes an asynchronous action with a message as parameter, it starts an asynchronous invocation. The action returns no value and the requesting object does not wait for a reply from the requested object. When the requesting object and the requested object are on different machines, the requesting one even does not wait until the other object actually receives the message.

On receiving the message, the host object might queue it according to a specific strategy, which is not in the scope of the basic action semantics.

If values are to be returned, the requested object must execute another asynchronous or synchronous action to sent them back.

▶ Synchronous messages

As a contrast, if the requesting object executes a syncronous action, it is blocked until a reply is received from the requested object. Even if there is no return value for the invocation, the requested object must send back a message indicating the processing is complete, and the requesting object is allowed to proceed with its jobs.

▶ Timing (1)

For sequential models or parts of models, the ordering of action executions conforms to the timing causality.

▶ Timing (1)

For sequential models or parts of models, the ordering of action executions conforms to the timing causality.

For concurrent models or parts of models, the ordering of action executions is unimportant, unless different action executions are required to synchronize to access shared variables. At other time, actions may be executed concurrently or in a meta-model-dependent order.

▶ Timing (2)

- The behavior of a model contains a number of *execution entities*, each of which has its own life cycle from the creation time till the destruction time.

▶ Timing (2)

- The behavior of a model contains a number of *execution entities*, each of which has its own life cycle from the creation time till the destruction time.
- An entity has an *identity*, which is unique and does not change over time.

▶ Timing (2)

- The behavior of a model contains a number of *execution entities*, each of which has its own life cycle from the creation time till the destruction time.
- An entity has an *identity*, which is unique and does not change over time.
- The dynamic behavior of an entity is a sequence of *snapshots*, each of which represents a stable state.

▶ Timing (2)

- The behavior of a model contains a number of *execution entities*, each of which has its own life cycle from the creation time till the destruction time.
- An entity has an *identity*, which is unique and does not change over time.
- The dynamic behavior of an entity is a sequence of *snapshots*, each of which represents a stable state.
- A *change* causes a transition between two snapshots of an entity.

▶ Timing (3)

- *Time* is an important property of a change, which specifies when a change occurs.

▶ Timing (3)

- *Time* is an important property of a change, which specifies when a change occurs.
- A sequence of changes of an entity over time constitutes a *history*, which starts from the creation of the entity and ends with the its destruction.

▶ Timing (4)

The change is placed in a history, which is a sequence of changes for an entity. The order of those changes conforms to the incremental order of their time attributes.

▶ Timing (4)

The change is placed in a history, which is a sequence of changes for an entity. The order of those changes conforms to the incremental order of their time attributes.

In this way, changes are carried out one after another, with the invariance that the first change (always at time 0) is the creation of the entity, and the last change its destruction.

▶ Discussion

Action semantics gives a better definition on the behavior of model execution, but there are still gaps left: i.e. the system-dependence of computational actions and the time for a change to take place.

▶ Discussion

Action semantics gives a better definition on the behavior of model execution, but there are still gaps left: i.e. the system-dependence of computational actions and the time for a change to take place.

It is still not standardized, and currently there are very limited materials on this subject. Some of the useful articles are listed in the reference.

▶ The next step

- Get more deeply into action semantics and find some successful concrete examples. I.e. the tools presented on the action semantics website `http://www.brics.dk/Projects/AS/`, like Actress, ASD Tools, OASIS and Recife Action Tools.

▶ The next step

- Get more deeply into action semantics and find some successful concrete examples. I.e. the tools presented on the action semantics website `http://www.brics.dk/Projects/AS/`, like Actress, ASD Tools, OASIS and Recife Action Tools.
- If a more detailed and precise manual is found, build a prototype of action semantics interpreter plug-in for the Statechart Virtual Machine (SVM) — with no much hope to complete in this term.

- **So much for today...**

Thank you for your attendance!

Any problems or concerns, please email: thomas@email.com.cn

References

[AILKC+00] Alcatel, I-Logix, Kennedy-Carter, Inc. Kabira Technologies, Inc. Project Technology, Rational Software Corporation, and Telelogic AB. *Action Semantics for the UML*. Document ad/2001-03-01. OMG, 2000. Available from World Wide Web: `http://cgi.omg.org/cgi-bin/doc?ad/01-03-01`.

[Mos96] Peter D. Mosses. Theory and practice of action semantics. In *MFCS '96, Proc. 21st Int. Symp. on Mathematical Foundations of Computer Science (Cracow, Poland, Sept. 1996)*, volume 1113, pages 37–61. Springer-Verlag, 1996. Available from World Wide Web: `http://www.brics.dk/RS/96/53/BRICS-RS-96-53.pdf`.

[Mos02] Peter D. Mosses. Action semantics and ASF+SDF - system demonstration, 2002. Available from World Wide Web: `http://www.cwi.nl/ftp/markvdb/entcs65.3/65.3.003.pdf`.

[MTAL98] Stephen J. Mellor, Steve Tockey, Rodolphe Arthaud, and Philippe LeBlanc. Software-platform-independent, precise action specifications for UML. In Jean Bézivin and Pierre-Alain Muller, editors, *The Unified Modeling Language, UML'98 - Beyond the Notation. First International Workshop, Mulhouse, France, June 1998*, pages 281–286, 1998. Available from World Wide Web: `http://www.kc.com/as_site/download/UML_AS_paper.pdf`.

[SGJ02] Gerson Sunyé, Alain Le Guennec, and Jean-Marc Jézéquel. Using UML action semantics for model execution and transformation. *Information Systems*, 27:445–457, 2002. Available from World Wide Web: `http://www.sciencedirect.com/science/article/B6V0G-45J8WV9-1/1/8622e2f14c9a518a5241431af02e27d8`.