Larissa: Semantic Aspects for Reactive Systems

David Stauch
VERIMAG, Grenoble, France

Motivation

- Crosscutting concerns pose problems in reactive, synchronous programs.
- Aspect-oriented programming seems a good candidate to address these.
- Existing aspect languages cannot be used: they lack necessary semantic properties and the underlying languages are very different.

Goal

- Develop an aspect-oriented extension for a synchronous language that modularizes recurrent crosscutting concerns of reactive systems.
- Aspects should integrate with the rest of the language, and have the usual semantic properties.

Context: Reactive Systems

Reactive systems are systems which continuously interact with their environment. They are often safety critical (e.g. in aircrafts or power plants). Therefore, they are usually programmed in dedicated languages which allow formal verification.

Synchronous Programming

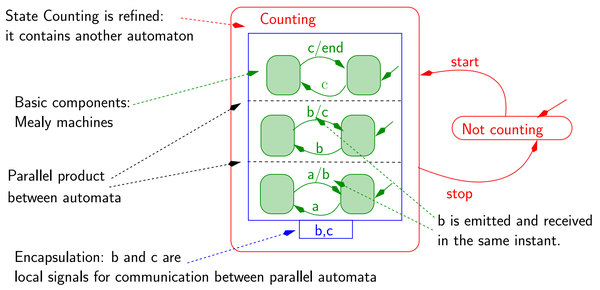
- Reactive systems are more naturally modeled as parallel units, but mostly executed on a single processor. Synchronous languages compile explicitly parallel programs into sequential code, and thus keep full control over execution.
- Time is discrete, and communication between parallel components is instantaneous, because entirely compiled.
- Synchronous languages include Esterel, Lustre, Signal, Safe State Machines, Argos.

Argos, a Simple Synchronous Language

- Argos is a StateChart-like hierarchical automata language, but with formally-defined synchronous semantics.
- It is the simplest synchronous language with a parallel structure, and thus a good choice as a base language for aspects.

An Example for Argos: 3-bit Counter

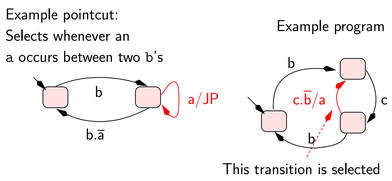
Inputs `start`, `stop`, and `a`, output `end`. `start` starts the counter, `stop` stops it, `end` is emitted every eight `a` while the counter is counting.



Larissa

Pointcut Language

- We want *semantic* aspects, that do not refer to the internals of the advised program.
- A pointcut is an automaton which observes the input and outputs of the base program, and emits a signal JP when the program is in a join point.
- A set of *join point transitions* is selected, by calculating a parallel product of the pointcut and the program and selecting those transitions that emit JP.

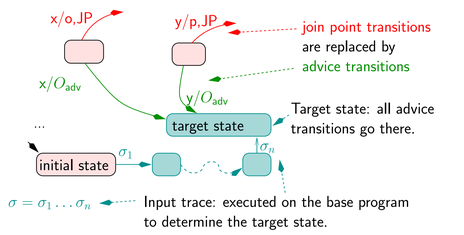


Advice

Larissa has three kinds of advice: toInit, toCurrent, and recovery advice (not explained here). A piece of advice replaces each join point transition by an advice transition, which has a different target states and different outputs.

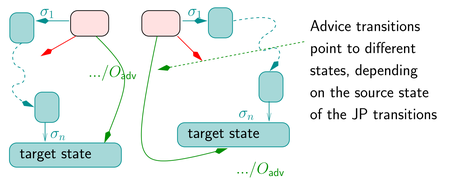
ToInit Advice

- Idea: jump to a fixed position in the program.
- The target state of the advice transitions is chosen by a finite input trace σ, executed from the initial state. The outputs of the advice transitions O_{adv} are globally defined by the advice.



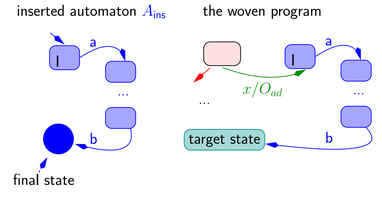
ToCurrent Advice

- Idea: jump forward from the join point.
- The target state is also chosen by a finite input trace, but it is executed from the source state of the join point transition.



Advice Transition vs Advice Program

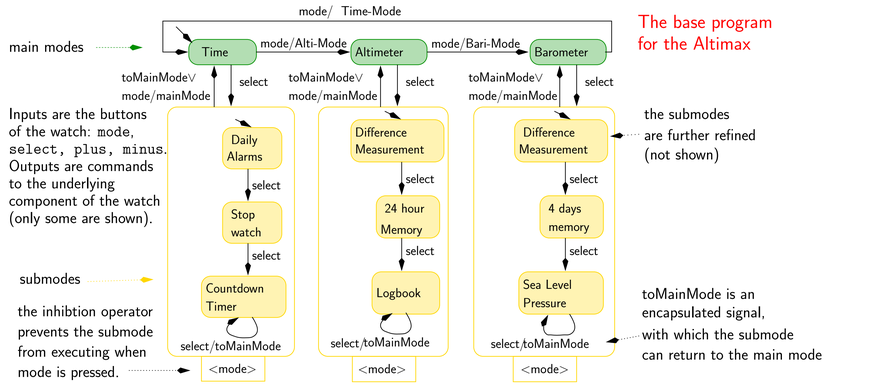
- Advice transition: join point transitions go directly to the target state (shown above).
- Advice program: specify an automaton A_{ins} with an initial state and a final state. The weaving inserts A_{ins} into the program, and advice transitions point to the initial of A_{ins}, and transitions of A_{ins} going to the final state are redirected to the target state in the woven program. A_{ins} must be inserted once per target state.



Aspect weaving preserves *trace equivalence:* when an aspect is woven into two trace-equivalent programs, the woven programs are still trace-equivalent.

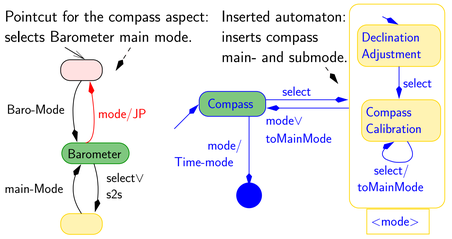
Case Study: Suunto Wristwatch

As an example, we model the interfaces of two complex wristwatches with several functionalities: watch, altimeter, barometer, and compass. In the interface, each functionality is represented by a main mode, which has several submodes. We model a product line with two models: Vector and Altimax (has no compass).



Building a Product Line: Altimax and Vector

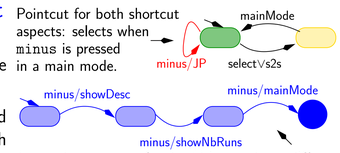
The Vector model contains a compass. We use an aspect to add the compass mode to the base program of the Altimax. The advice is implemented with a toInit advice with target state Time main mode (empty trace), and an advice program which contains the main mode and the submodes for the compass.



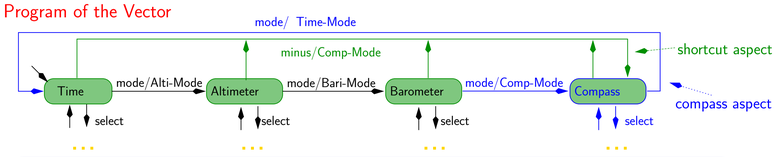
Encapsulating Crosscutting Concerns: Shortcut Aspects

Pressing `minus` in the main modes has different functions in the different models:

- The Altimax shows information from the altimeter logbook and returns then to the main mode in which it was. Implemented with a toCurrent advice, an empty trace and an advice program.
- The Vector goes directly to the Compass mode. Implemented with a toInit advice with trace `mode.mode.mode.mode`.



Woven Program of the Vector



Evaluation of the Case Study

- Larissa encapsulates crosscutting concerns (shortcut aspects).
- Larissa offers the possibility to re-use code to build several models.

Bibliography

[1] sketched some ideas, [2] introduces and formally defines Larissa, [3] details the case study, and [5] analyzes aspect interference. An prototype implementation for Larissa can be found at [4].

[1] K. Altisen, F. Maraninchi, and D. Stauch. Exploring aspects in the context of reactive systems. In *Workshop on the Foundations of Aspect-Oriented Languages (FOAL)*, March 2004.

[2] Karine Altisen, Florence Maraninchi, and David Stauch. Aspect-oriented programming for reactive systems: a proposal in the synchronous framework. *Science of Computer Programming, Special Issue on Foundations of Aspect-Oriented Programming*, 2006. To appear.

[3] Karine Altisen, Florence Maraninchi, and David Stauch. Larissa: Modular design of man-machine interfaces with aspects. In *5th International Symposium on Software Composition*, Vienna, Austria, March 2006. To appear.

[4] Compiler for Larissa. http://www-verimag.imag.fr/~stauch/ArgosCompiler/.

[5] D. Stauch, K. Altisen, and F. Maraninchi. Interference of Larissa aspects. In *Foundations of Aspect-Oriented Languages (FOAL)*, March 2006.