Implementation of advanced SCOOP aspects

PROJECT PLAN

Master Thesis	Implementation of advanced SCOOP aspects
Project period	26.10.2010 – 26.4.2010
Student name	Damien Müllhaupt
Email address	damienm@student.ethz.ch
Supervisor name	Benjamin Morandi

1. PROJECT DESCRIPTION

Overview

In todays software development concurrency plays a major role and demands close attention. Due to the complexity of the matter, concurrency demands a great deal of thoughts and attention from the developer and is hard to get right. Simple concurrent object oriented programming SCOOP provides a simple object oriented model to deal with concurrency.

The basic principles of SCOOP are described in the doctoral thesis from Piotr Nienaltowski[4]. As part of his thesis, a first standalone implementation of SCOOP was creaed. The implementation was refactored and extended as a part of the Eiffel Verification Environment called EVE, a research branch of EiffelStudio. This re-implementation was done in the master thesis from Patrick Huber. Due to time constraints some advanced aspects were only partially implemented leaving the need of expanding and revising the current implemantation to include those advanced aspects.

Scope of the work

The goal of this project is to improve the new implementation of SCOOP in EVE and to add the missing aspects. These aspects include:

1) Support for feature re-declaration

    a. The current implementation does not correctly process code with feature re-declarations involving separate arguments or separate return types. The following example explains the problem. The following is valid SCOOP code:

```
class
    A

feature
    f (a: X): separate X
            do
                    ...
            end
end

class
    B

inherit
    A
            redefine f end

feature
    f (a: separate Y): Y
            do
                    ...
            end
end

class
    X
end

class
    Y

inherit
    X
end
```

This code leads to the generation of a client class similar to:

```
class
   A

feature
   f (a: X): SCOOP_SEPARATE_X
           do
                   ...
           end
end

class
   B

inherit
   A
           redefine f end

feature
   f(a: SCOOP_SEPARATE_Y): Y
           do
                   ...
           end
end
```

The generated code is not valid Eiffel code, because it does not follow the Eiffel redefinition rules. The class SCOOP_SEPARATE_Y is not a subtype of X and Y is not a subtype of SCOOP_SEPARATE_X.

2) Support for separate upcasts and downcasts

   a. The overall goal is to make sure that our processed code supports upcasts and downcasts in the presence of proxy and client objects. To be more precise, we need to support code of the following kind where B inherits from A:

```
a_sep: separate A
b: B
create b

a_sep := b -- Upcast
if attached {l: separate B} a_sep then
      -- Downcast from separate to separate.
end
if attached {l: B} a_sep then
      -- Downcast from separate to non-separate.
end
```

3) Implementation of agent mechanism

   a. Agents have to be created on the processor where the target is residing. In the current build of SCOOP they are created on the processor the caller resides on.

4) Implementation of exception mechanism
5) Implementation of import mechanism

The time used in each individual point is hard to predict. Each point is subject of change depending on how much time is left after completing the previous task. This demands a rather flexible schedule adapting to the needs and problems arising during the implementation of those tasks.

Intended results

The intention is to implement those advanced SCOOP aspects in EVE. The general idea is to implement and finish as many advanced aspects as possible prioritized by the list given in the next section.

2. BACKGROUND MATERIAL

Reading List

See reference section.

3. PROJECT MANAGEMENT

Objectives and priorities

The goal is to implement the advanced SCOOP aspects with respect to the following list of priorities:

1) Support for feature re-declaration
2) Support for separate upcasts and downcasts
3) Implementation of agent mechanism
4) Implementation of exception mechanism
5) Implementation of import mechanism

Criteria for success

The project is a success when the first three topics are successfully implemented, tested and documented.

Method of work

Weekly meetings will be held to report on the progress of the project and to make decisions about next steps. Continuous supervisor feedback is expected.

Collaboration with the other members of the SCOOP team is necessary, because of the ongoing work on the SCOOP compiler and the SCOOP library.

Quality management

Documentation

A complete and rich documentation of the implementation has to be provided in order to support further work on the implementation of SCOOP in EVE.

Validation steps

The validation is supported by regular meetings. Continuous code reviews and testing are important in order to validate the implementation.

4. PLAN WITH MILESTONES

Project steps

22.10.2009: Meeting with Betrand Meyer.

26.10.2009: Start of the master thesis.

26.10.2009-26.12.2009 Implementation of the first criteria.

26.12.2009-26.01.2009 Implementation of the second criteria.

26.01.2009-26.02.2009 Implementation of the third criteria.

26.02.2009-26.03.2009 Implementation of the fourth and fifth criteria.

26.03.2009-26.04.2009: Documentation and composition of the master thesis

Deadline

The deadline is set to the 26th of April 2010.

REFERENCES

[1]. Chair of Software Engineering: *Semester- /Diplomarbeiten*; Online at: http://se.inf.ethz.ch/projects/index.html, consulted in October 2002.

[2]. Bertrand Meyer: Object-Oriented Software Construction, 2nd edition, Prentice Hall, 1997.

[3]. Eiffel Verification Environment (EVE) http://eve.origo.ethz.ch/

[4]. Piotr Nienaltowski, Practical framework for contract-based concurrent object-oriented programming, http://se.inf.ethz.ch/people/nienaltowski/papers/thesis.pdf

[5]. Patrick Huber, Integrating SCOOP into EVE, http://se.inf.ethz.ch/projects/patrick_huber