

SCOOP project: SANTA STRIKES BACK

Hand-out: 23 May
Due: 4 July

The goal of the project is to design and implement two interesting synchronisation scenarios in SCOOP. No GUI is required (hooray!) – the applications can use either the standard text output (console) or a text log file. The main focus is on the concurrency issues and the way you handle them with SCOOP. You can solve the exercise in teams of up to three people (hooray again!).

*Please submit the source code of your applications (a zip file) and a project report by e-mail to piotr.nienaltowski@inf.ethz.ch until **July 4**. You will need to present your project to the assistant before July 18.*

1. Santa Claus

Surprise, surprise! You did not expect this one, did you? This synchronisation scenario has already served as homework: we have tried to tackle the problem using Java and multithreading. Let's recall the problem statement.

Santa lives in his little house in Jjävaskjølgbrø; his favourite activity is to have a nap. So he sleeps all the time, except for rare moments when he is awoken by either a group of elves or a group of reindeer. There are 10 elves who live in the area and work in a toy workshop. They produce toys and, once in a while, they run out of ideas and need to ask Santa for help. They can only wake up Santa if they form a group of three. There are also nine reindeer that live in the stable nearby; Santa uses them to deliver toys to kids. The reindeer are as lazy as Santa himself, so for the most of the day they just sleep. When they want to work, they need to form a group of nine (that is all the reindeer must join) and wake up Santa. If they manage to do it, Santa does the delivery round and then goes back to sleep (so do the reindeer). It is important to note that in a situation when there is a group of three elves and the group of nine reindeer trying to wake up Santa at the same time, it is the reindeer that get the priority.

This time, you have to implement the scenario in SCOOP. Your application should be based on classes *SANTA*, *REINDEER*, *ELF*, plus any additional classes that implement useful abstractions. Use inheritance whenever necessary – remember that abstraction is beautiful and there is nothing uglier than an application with several classes that look almost the same and contain identical code.

2. Active objects

The goal of this task is to find an elegant and efficient way to simulate active objects in SCOOP. SCOOP does not directly support the concept of active object but we should be able to implement scenarios that require the use of rendezvous synchronisation. Our beloved Santa has agreed to serve as example once again.

Santa has finally decided to go with the wind of change and make his business Internet-based. Before going online, he wants to try out locally a simple client-server approach to optimise his consultancy activities. From now on, the elves who want to consult Santa are not required to form a group anymore – they may simply send an individual request to Santa’s server and wait until Santa accepts to meet them using a new teleconferencing support. After issuing a request, an elf is blocked until Santa has accepted his request and met him. After the consultation the elf should work on his own for a while, then play with his kids (each elf has three kids; the kids are always ready to play with their dad, except when they are eating). Important: elves never sleep, they are always active!

Unlike elves, Santa sleeps most of the time. Nevertheless, he wakes up once in a while to check if there are any requests from the elves. If there are any pending requests, he accepts a request and consults the elf who issued the request. Santa does not need to process all pending requests – in fact, we should be able to parametrise Santa to accept at most n requests in a row. After satisfying a given number of requests Santa goes back to sleep. A very important thing: the old Santa is able to process just one request (that is meet just one elf) at a time!

Oh, one more thing: Santa has optimised the business and he only does consulting; UPS has taken over his delivery business, so no reindeer are involved anymore. They were useless anyway, weren’t they?

Your task is to implement the scenario in SCOOP. Obviously, the client-server scenario described above calls for a rendezvous-style synchronisation. You need to implement active objects that represent Santa, elves, and elves’ kids. Make sure that their schedules (bodies) specify correctly their activities:

- *SANTA*: sleep, accept some pending request, process the request (i.e. meet the elf), and so on. Remember that accepting a request is a blocking activity.
- *ELF*: issue a request to Santa, work a bit on your own, play with your kids. Remember that after issuing a request the elf is blocked until Santa accepts the request. Also, the elf has to play with all his kids (not necessarily at the same time) before issuing the next request to Santa.
- *KID*: eat, play with your dad, eat, play with your dad, and so on.

Once again, make sure that you make an appropriate use of inheritance to achieve the necessary level of abstraction. Try to provide a general class (or set of classes) that implements an active object and then derive specialised classes for Santa, elves, and kids. Try to parametrise Santa in such a way that he accepts *at most n* pending requests before going back to sleep.

Sleeping, elves’ own work, eating, and playing can be represented with simple routines *nap*, *work*, *eat*, and *play*. The details are irrelevant – it is enough to output a corresponding message, e.g. “Santa is sleeping”.

3. Assessment

The final mark will depend on the following criteria:

- Correct implementation: the applications do what they are supposed to do.
- Proper use of O-O mechanisms.
- Quality of the accompanying project report.

Questions to be answered in the project report:

- 3.1. What synchronisation patterns (producer-consumer, reader-writer, etc.) have you used in each application?
- 3.2. How have you implemented each pattern? Give a short description of the algorithm you have used and include code snippets.
- 3.3. What are the main problems with the implementation of these synchronisation patterns in SCOOP?
- 3.4. Can you think of additional mechanisms or abstractions that SCOOP should offer?
- 3.5. As a programmer, what is your feeling about SCOOP? Does it allow you to program efficiently and express what you want to express? How would you compare SCOOP with other mechanisms (say Java multithreading with monitors) in terms of expressivity, usability and “programmer-friendliness”? When answering that question, please try to consider all important elements, e.g. ease of code reuse, etc.
- 3.6. What is your opinion on scoop2scoopli? How would you improve the tool?

4. Tools

You will use the EiffelStudio 5.5 IDE, together with the scoop2scoopli pre-processor.

5. Support

You have the possibility to ask questions during the exercise sessions on Tuesdays. You can also send questions by e-mail (scoop-support@se.inf.ethz.ch). Additionally, you can pop in at RZ J3 during office hours (from 16:00 to 17:30 on Tuesdays) . If you need an individual meeting with me on a different day, just send a request to scoop-support@se.inf.ethz.ch.