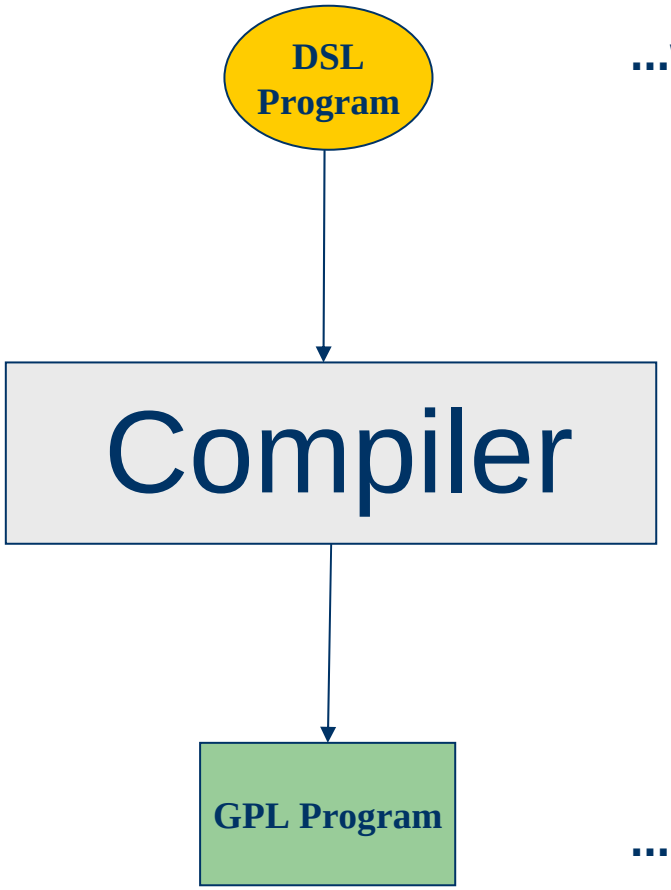
AOP to Simplify DSL Implementation

Laurent Réveillère

IFIP 2.11 meeting

Jan. 26, 2006

Challenges in DSL Compilation



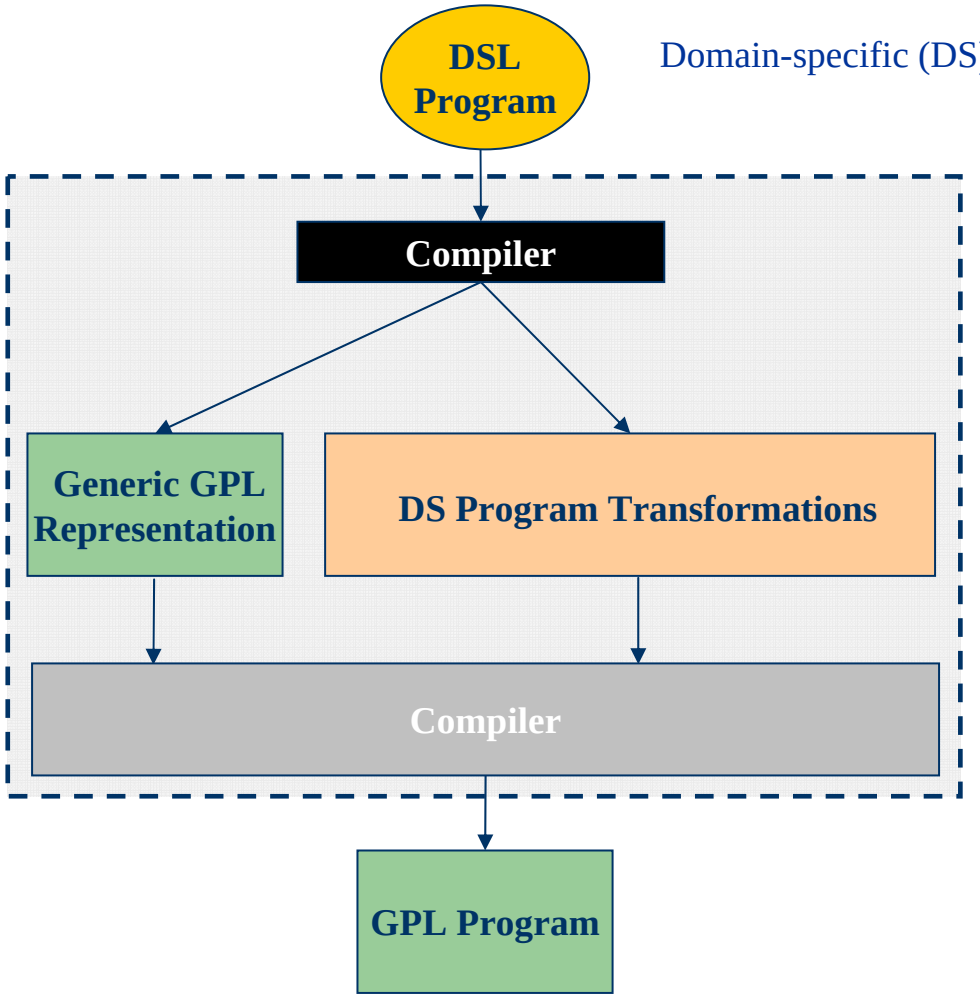
...what to compute

DSL compilation deals with:

- ✓ Program logic
- ✓ State management
- ✓ Resource management
- ✓ Mapping to underlying software layers
- ✓ Analyses
- ✓ Optimizations
- ✓ ...

... how to compute

Our Approach

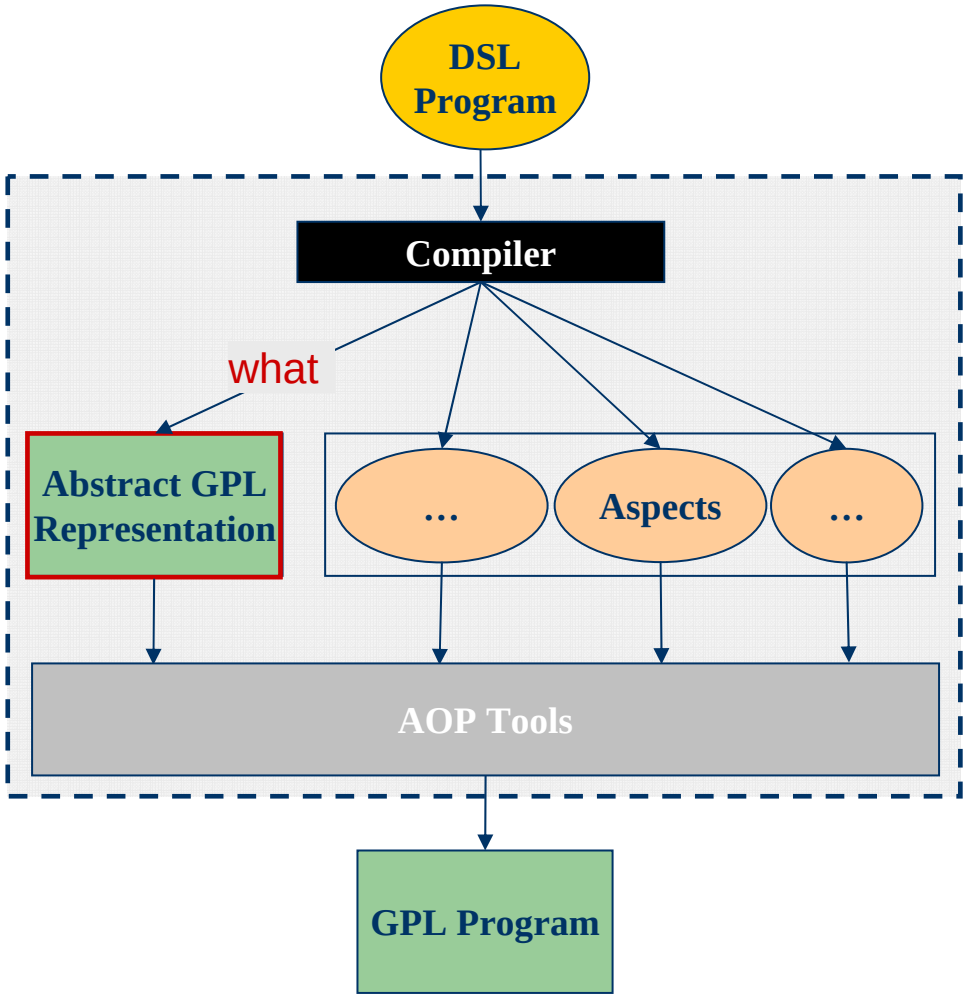


Standard DSL compilation :

- Traditional compilation of non DS information
- Compilation of DS information into DS program transformations

=> Key point: processing of DS information

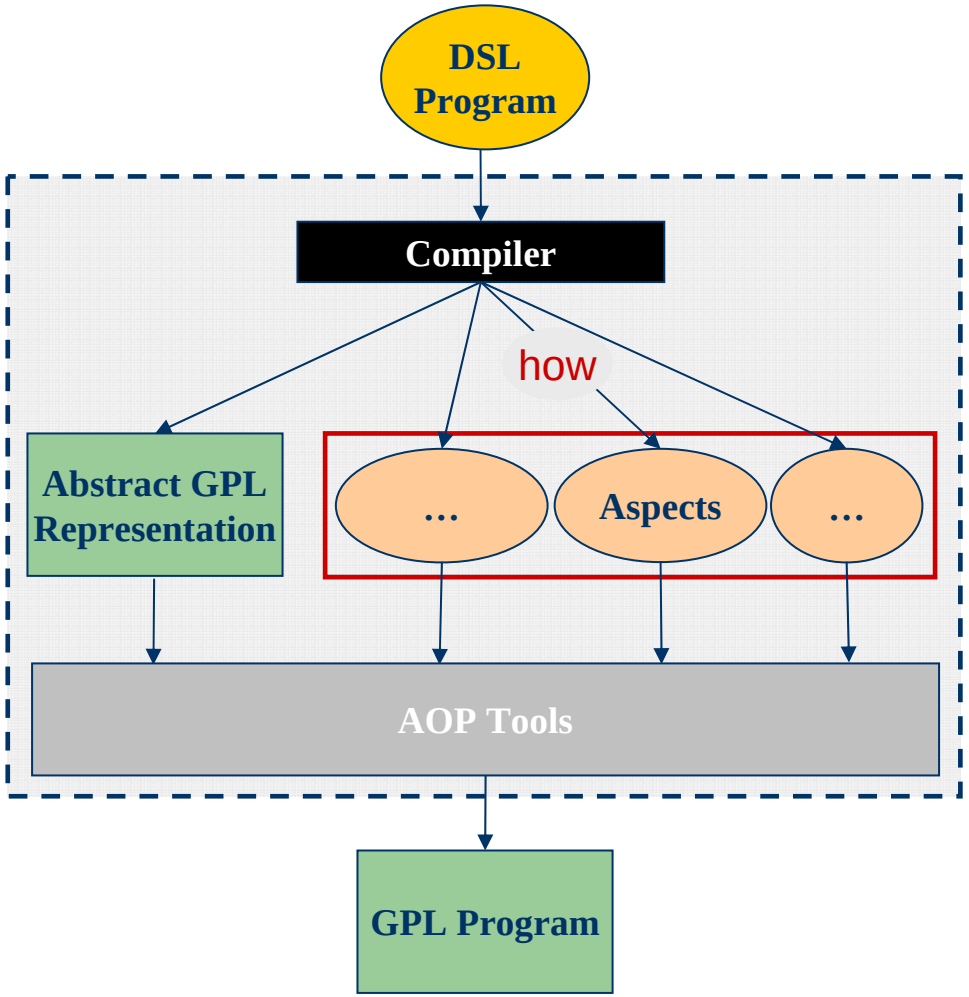
Our Approach



- ✓ Only program logic translation,
- ✓ Partial interpretation,
- ✓ Domain-specific information encoding,
- ✓ Program structuring.

=> Amenable to AOP tools.

Our Approach



✓ Specific implementation details related to a concern

=> Compilation is now viewed through different implementation concerns

Case study : SPL Compiler

```
service Counter {
  processing {
    local void log (int);
    registration {
      int count;
      response outgoing REGISTER() {
        count = 0;
        return forward;
      }

      void unregister() {
        log (count);
      }

      dialog {
        response incoming INVITE() {
          response resp = forward;
          if (resp != /SUCCESS) {
              return forward 'sip:secretary@company.com';
          } else {
            count++;
            return resp;
  } }}}}
```

DSL

GPL

```
public class Counter implements SipListener {
  [...]
  private void handler_REGISTER (Request rq, String met



         count = 0;
                  sendRequest (false, rq_request);
  }
  private void handler_unregister (Request rq, String m


         log (state.count);

                  sendRequest (false, rq_request);
  }
  private void handler_INVITE (Request rq, String metho



      sendRequest (true, rq_request);
  }
```

```
public void processRequest (RequestEvent requestEvent) {


    String method = rq_request.getMethod();
    if (method.equals (Request.REGISTER)) {
      if (!registrar.hasExpiresZero (rq_request)) {
        if (!registrar.hasRegistration (rq_request)) {
          handler_REGISTER (rq_request, method);
        } else {
                        sendRequest (false, rq_request);
        }
      } else if (registrar.hasRegistration (rq_request)) {
        handler_unregister (rq_request, method);
      } else {
                      sendRequest (false, rq_request);
      }
    } else if (method.equals (Request.INVITE)) {
      handler_INVITE (rq_request, method);
    } else {
                    sendRequest (false, rq_request);
    }
  }
```

```
public void processResponse (ResponseEvent responseEvent) {









      String method = rs_request.getMethod();
      rs_responseCode = rs_response.getStatusCode();
      if (method.equals (Request.INVITE)) {
        if (rs_responseCode >= 300) {
          AddressFactory addressFactory = getAddressFactory();
          SipURI sipURI = addressFactory.createSipURI ("secretary",
"company.com");
          rs_request.setRequestURI (sipURI);
                        sendRequest (false, rs_request);
        } else {
               count++;



             sendResponse (true, rs_response);

        }
      } else {



                      sendResponse (false, rs_response);
      }



                      sendResponse(rs_response);


  }

  public void processTimeout (TimeoutEvent timeoutEvent) {[...]}
}
```

Case study : SPL Compiler

```
service Counter {
  processing {
    local void log (int);
    registration {
      int count;
      response outgoing REGISTER() {
        count = 0;
        return forward;
      }

      void unregister() {
        log (count);
      }

      dialog {
        response incoming INVITE() {
          response resp = forward;
          if (resp != /SUCCESS) {
              return forward 'sip:secretary@company.com';
          } else {
            count++;
            return resp;
  } }}}}
```

DSL

GPL

```
public class Counter implements SipListener {
  [...]
  private void handler_REGISTER (Request rq, String met
    State state = new State();
    int ident = lib.env.getId(rq);
    lib.env.setEnv(ident, state);
    state.count = 0;
    rq_sipProvider.sendRequest (false, rq_request);
  }
  private void handler_unregister (Request rq, String m
    int ident = lib.env.getId(rq);
    State state = lib.env.getEnv(ident);
    local.log (state.count);
    lib.env.delEnv(ident);
    rq_sipProvider.sendRequest (false, rq_request);
  }
  private void handler_INVITE (Request rq, String metho
    int ident = lib.env.getId(rq);
    State state = lib.env.getEnv(ident);
    ClientTransaction ct =
rq_sipProvider.getNewClientTransaction(rq);
    ct.sendRequest (true, rq_request);
  }
```

```
public void processRequest (RequestEvent requestEvent) {
    Request rq_request = requestEvent.getRequest();
    SipProvider rq_sipProvider = (SipProvider)
requestEvent.getSource();
    String method = rq_request.getMethod();
    if (method.equals (Request.REGISTER)) {
      if (!registrar.hasExpiresZero (rq_request)) {
        if (!registrar.hasRegistration (rq_request)) {
          handler_REGISTER (rq_request, method);
        } else {
          rq_sipProvider.sendRequest (false, rq_request);
        }
      } else if (registrar.hasRegistration (rq_request)) {
        handler_unregister (rq_request, method);
      } else {
        rq_sipProvider.sendRequest (false, rq_request);
      }
    } else if (method.equals (Request.INVITE)) {
      handler_INVITE (rq_request, method);
    } else {
      rq_sipProvider.sendRequest (false, rq_request);
    }
  }
```

```
public void processResponse (ResponseEvent responseEvent) {
    int ident = lib.env.getId(rs_request);
    State state = lib.env.getEnv(ident);
    ClientTransaction rs_ct = responseEvent.getClientTransaction();
    if (rs_ct != null) {
      Request rs_request = rs_ct.getRequest();
      Response rs_response = responseEvent.getResponse();
      SipProvider rs_sipProvider = (SipProvider)
responseEvent.getSource();
      String method = rs_request.getMethod();
      rs_responseCode = rs_response.getStatusCode();
      if (method.equals (Request.INVITE)) {
        if (rs_responseCode >= 300) {
          AddressFactory addressFactory = getAddressFactory();
          SipURI sipURI = addressFactory.createSipURI ("secretary",
"company.com");
          rs_request.setRequestURI (sipURI);
          rs_sipProvider.sendRequest (false, rs_request);
        } else {
          state.count++;
          try {
            ClientTransaction ct =
responseEvent.getNewClientTransaction();
            ct.sendResponse (true, rs_response);
          } catch (Exception ex) {}
        }
      } else {
        if (rs_responseCode >= 300) {
          lib.env.delEnv(ident);
        }
        rs_sipProvider.sendResponse (false, rs_response);
      }
    } else {
      try {
        rs_sipProvider.sendResponse(rs_response);
      } catch (Exception ex) {}
    }
  }

  public void processTimeout (TimeoutEvent timeoutEvent) {[...]}
}
```

Conclusion

- Advantages
  - Modular treatment of domain-specific information
  - Generated aspects make AOP approach predictable
    - » Does not break the semantic of the program

- Remaining issues
  - Structuring the abstract GPL representation is not so easy
  - Limitations of the pointcut language

➔Is AOP expressive enough for DSL implementation?