# SDL Modules – Concepts and Tool Support

Philipp Becker, Marc Krämer
{pbecker,kraemer}@cs.uni-kl.de

http://vs.cs.uni-kl.de

04.10.2010 – SAM 2010, Oslo, Norway

## Outline

## SDL – Overview

### Advantages of SDL

- Hierarchical, modular structure
- Different abstraction levels
- Platform-independent specifications
- Well-defined semantics
- Reuse of components
  - Dependency relations
  - Black-box behavior
- Inheritance

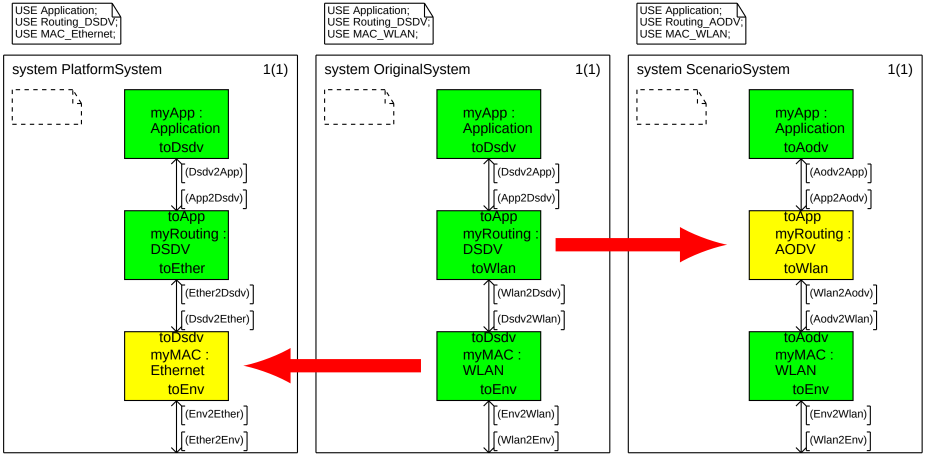
### Disadvantages of SDL

- Complexity correlates with system size (not limited to SDL)
- Inheritance mechanism difficult to use
- Systems not always platform-independent

### Tool Shortcomings

- Missing refactoring support

## System Adaptation



Platform-based adaptation

Original System

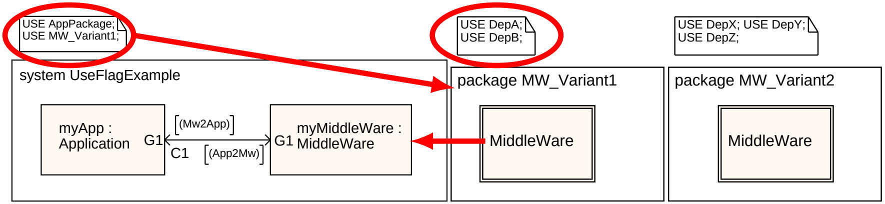
Scenario-based adaptation

## System maintenance

**Question**

How to maintain **one** system for different platforms / scenarios?

- Exchange of SDL packages via `USE` statements
  - Block types, process types, service types
  - Data types, signals, other definitions
  - Syntactical compatibility required for types



## System maintenance

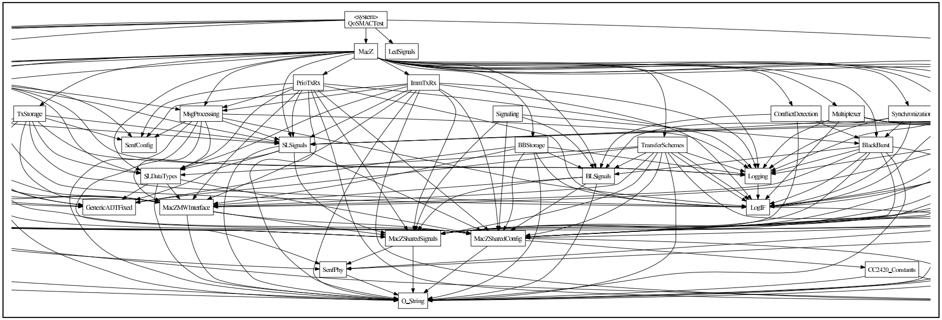
**Question**

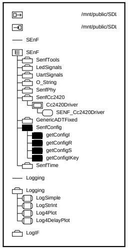
How to maintain **one** system for different platforms / scenarios?

- Exchange of SDL packages via `USE` statements
  - Block types, process types, service types
  - Data types, signals, other definitions
  - Syntactical compatibility required for types

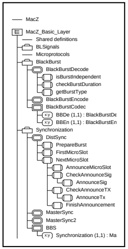
+ Maintain system structure
+ Changes only done on top level of packages
– Modify `USE` statements in multiple packages
– Dependencies still need to be managed

## Conceptual treatment & tool support

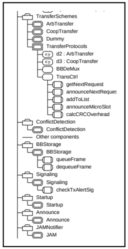




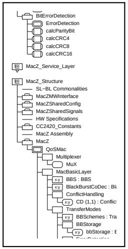
/mnt/public/SDL
/mnt/public/SDL
SEnF
SEnF
SenfTools
LedSignals
UartSignals
O_String
SenfPhy
SenfCc2420
Cc2420Driver
SENF_Cc2420Driver
GenericADTFixed
SenfConfig
getConfigI
getConfigR
getConfigS
getConfigKey
SenfTime
Logging
Logging
LogSimple
LogStrInt
Log4Plot
Log4DelayPlot
LogIF



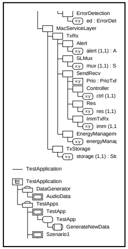
MacZ
MacZ_Basic_Layer
Shared definitions
BLSignals
Microprotocols
BlackBurst
BlackBurstDecode
isBurstIndependent
checkBurstDuration
getBurstType
BlackBurstEncode
BlackBurstCodec
BBDe (1,1) : BlackBurstDe
BBEn (1,1) : BlackBurstEn
Synchronization
DistSync
PrepareBurst
FirstMicroSlot
NextMicroSlot
AnnounceMicroSlot
CheckAnnounceSig
AnnounceSig
CheckAnnounceTX
AnnounceTx
FinishAnnouncement
MasterSync
MasterSync2
BBS
Synchronization (1,1) : Ma



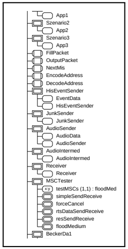
TransferSchemes
ArbTransfer
CoopTransfer
Dummy
TransferProtocols
d2 : ArbTransfer
d3 : CoopTransfer
BBDeMux
TransCtrl
getNextRequest
announceNextReques
addToList
announceMicroSlot
calcCRCOverhead
ConflictDetection
ConflictDetection
Other components
BBStorage
BBStorage
queueFrame
dequeueFrame
Signaling
Signaling
checkTxAlertSig
Startup
Startup
Announce
Announce
JAMNotifier
JAM



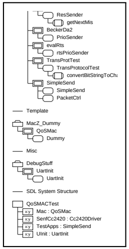
BitErrorDetection
ErrorDetection
calcParityBit
calcCRC4
calcCRC8
calcCRC16
MacZ_Service_Layer
MacZ_Structure
SL−BL Commonalities
MacZMWInterface
MacZSharedConfig
MacZSharedSignals
HW Specifications
CC2420_Constants
MacZ Assembly
MacZ
QoSMac
Multiplexer
MuX
MacBasicLayer
BBS : BBS
BlackBurstCoDec : Bl
ConflictHandling
CD (1,1) : Conflic
TransferModes
BBSchemes : Tra
BBStorage
bbStorage : E



ErrorDetection
ed : ErrorDet
MacServiceLayer
TxRx
Alert
alert (1,1) : A
SLMux
mux (1,1) : S
SendRecv
Prio : PrioTxR
Controller
ctrl (1,1)
Res
res (1,1)
ImmTxRx
imm (1,1
EnergyManagem
energyManag
TxStorage
storage (1,1) : Sto
TestApplication
TestApplication
DataGenerator
AudioData
TestApps
TestApp
TestApp
GenerateNewData
Szenario1



App1
Szenario2
App2
Szenario3
App3
FillPacket
OutputPacket
NextMis
EncodeAddress
DecodeAddress
HisEventSender
EventData
HisEventSender
JunkSender
JunkSender
AudioSender
AudioData
AudioSender
AudioIntermed
AudioIntermed
Receiver
Receiver
MSCTester
testMSCs (1,1) : floodMed
simpleSendReceive
forceCancel
rtsDataSendReceive
resSendReceive
floodMedium
BeckerDa1



ResSender
getNextMis
BeckerDa2
PrioSender
evalRts
rtsPrioSender
TransProtTest
TransProtocolTest
convertBitStringToCh
SimpleSend
SimpleSend
PacketCtrl
Template
MacZ_Dummy
QoSMac
Dummy
Misc
DebugStuff
UartInit
UartInit
SDL System Structure
QoSMACTest
Mac : QoSMac
SenfCc2420 : Cc2420Driver
TestApps : SimpleSend
UInit : UartInit

## Idea

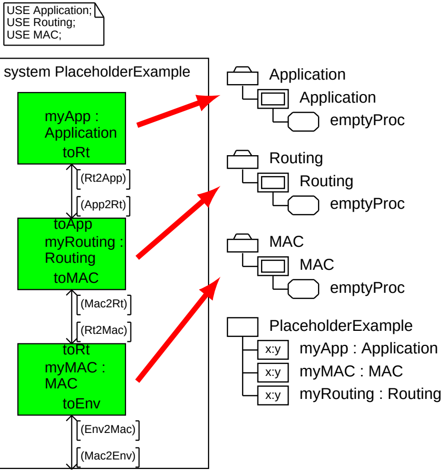
**Idea**

Introduce a concept to exchange arbitrary syntactically compatible packages.

- Based on existing SDL language features
- Utilize SDLs advantages
  - Modular structure
  - Black-box property
- Consider dependency relations
- Provide flexible tool support

## Alternative Application Scenario

- Build high level system by use of placeholders
  - Keep system as small as necessary
  - Develop components separately
  - Rapid development of systems from existing components



## SDL Module Concept

## Compatibility of SDL Packages

### Syntactical Compatibility

Defining the syntactical compatibility of SDL packages means defining the syntactical compatibility of their contents.

| Type | Requirement |
|---|---|
| Signals, Signal lists | Equal definitions |
| Data types | Equal set of operators |
| Synonyms | Compatible data types |
| Procedures | Equal signature |
| (Block/Process/Service) types | – In- and outgoing signals<br>– Gate names<br>– Correlation between both |

## SDL Module Interface

**Definition (SDL Module Interface)**

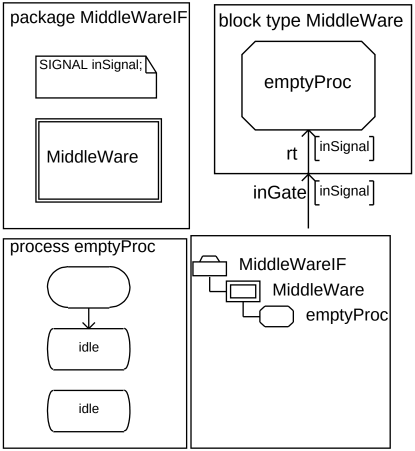
- Single SDL package
- Contains type definitions
  - Signals, Signal lists
  - Data types
  - Synonyms
  - Procedures
  - Block types, Process types, Service types
- Syntactically complete
- Definitions omit as much functionality as possible
- May contain informal SDL comments, e.g. to describe the intended usage / behavior

## SDL Module Interface – Example

```
PACKAGE MiddleWareIF;
  SIGNAL inSignal;
  BLOCK TYPE MiddleWare;
    GATE inGate IN WITH inSignal;
    SIGNALROUTE rt FROM env TO
        emptyProc WITH inSignal;

    PROCESS emptyProc;
      start;
      NEXTSTATE idle;
      STATE idle;
    ENDPROCESS emptyProc;

  ENDBLOCK TYPE MiddleWare;
ENDPACKAGE MiddleWareIF;
```

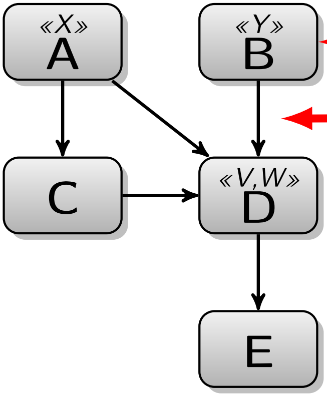


## SDL Module

**Definition (SDL Module)**

- Non-empty set of SDL packages
- Forms a transitive closure w.r.t. the packages dependencies
- Specifies or complies to an SDL Module Interface $X$ iff the Module contains an SDL package that is compatible to this interface
- May comply to arbitrary number of SDL Module interfaces
- Acts as exchangeable unit w.r.t. the used interfaces

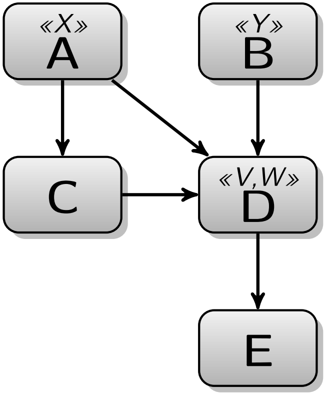
## SDL Modules – Example



Package B, complies to interface *Y*

USE dependency
(B depends on D)

## SDL Modules – Example



- $M_1 = \{A, B, C, D, E\}$ $(X, Y, V, W)$
- $M_2 = \{A, C, D, E\}$ $(X, V, W)$
- $M_3 = \{B, C, D, E\}$ $(Y, V, W)$
- $M_4 = \{B, D, E\}$ $(Y, V, W)$
- $M_5 = \{C, D, E\}$ $(V, W)$
- $M_6 = \{D, E\}$ $(V, W)$
- $M_7 = \{E\}$ $(\emptyset)$
- ~~$M_8 = \{A, B, C, D\}$ $(X, Y, V, W)$~~

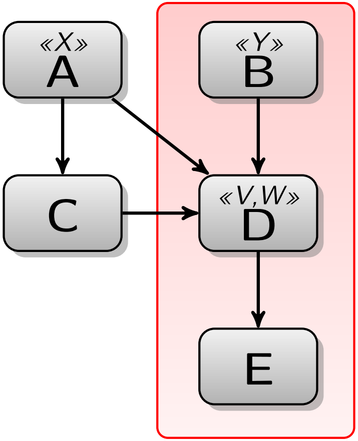
## Minimal SDL Modules

**Definition (Minimal SDL Module)**

- Regular SDL Module
- Minimal for a given SDL Module Interface $X$ iff...
  1. one SDL package $p$ in the module is compatible to $X$
  2. all other packages in the module are direct or indirect dependencies of $p$

- Module can still comply to other interfaces
- May not be minimal for these other interfaces
- **Minimal SDL Module for interfaces can be automatically derived**

## Minimal SDL Modules – Example



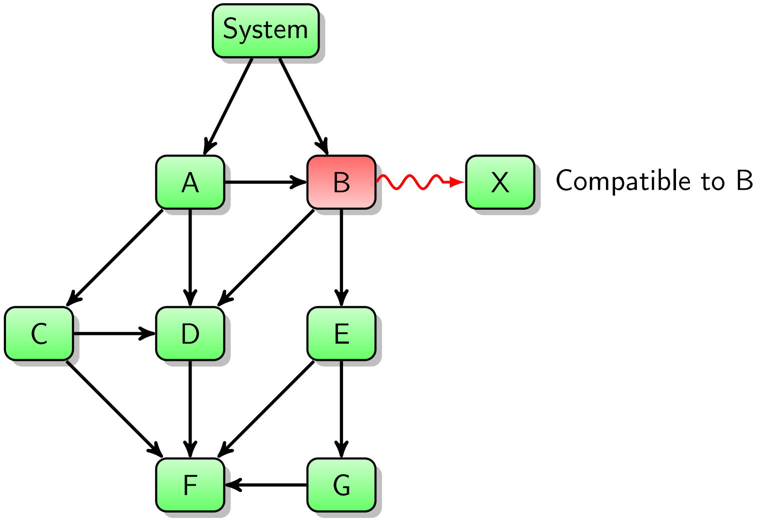
**Intention**

Automatically derive minimal SDL module that complies to interface $Y$

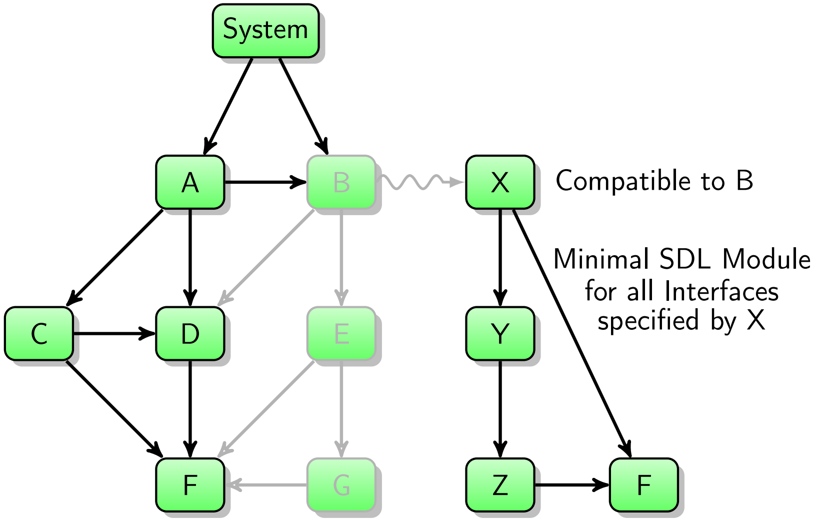
1. Start with package B
2. Include dependency D
3. Include dependency E

- $M_4 = \{B, D, E\}$ $(Y, V, W)$
  - Minimal for interface $Y$
  - **Not** minimal for interfaces $V, W$

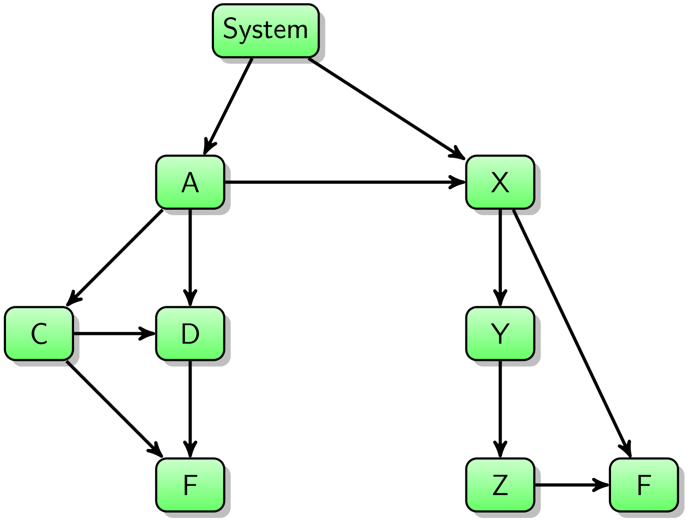
## Package Exchange with SDL Modules



## Package Exchange with SDL Modules



## Package Exchange with SDL Modules



## Package Exchange with SDL Modules

$$\begin{tikzcd}
 & \text{System} \arrow[d] \arrow[dr] & \\
 & A \arrow[r] \arrow[dl] \arrow[d] & X \arrow[ddl] \arrow[d] \\
C \arrow[r] \arrow[dr] & D \arrow[d] & Y \arrow[d] \\
 & F & Z \arrow[l]
\end{tikzcd}$$

## SDL Package Substitution Tool
## SPaSs

## SPaSs Overview

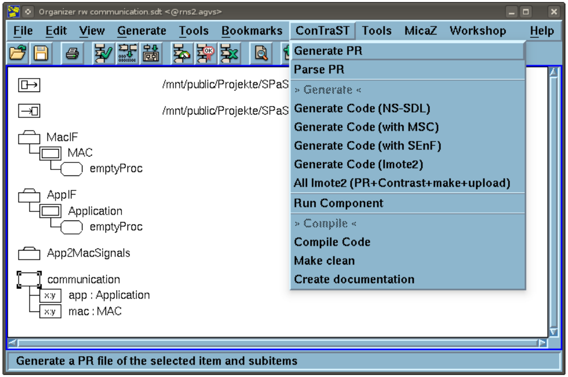
- SPaSs – SDL Package Substitution Tool
- Implementation of SDL Module concept
- Command-line tool, works on SDL/PR files
- Flexible configuration
- Platform-independent (Java)
- No syntax or interface checks so far
- Tested with PR code generated from Tau

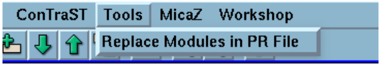
## Configuration

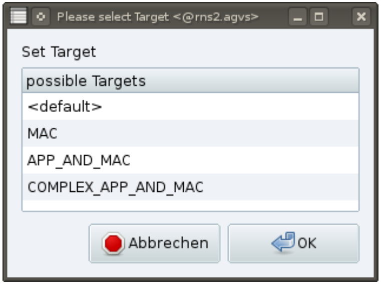
- XML-Files
- Configuration on different levels (System, User, Project)
- Definition of data sources to aquire SDL Packages
  - Directories
  - Version Control Repositories (Subversion, Git)
- Definition of mapping pairs $(p_{old}, p_{new})$
  - Grouped in (named) mapping sets (a.k.a. *targets*)
  - Choose dynamically, e.g. according to target platform

```
<Mapping>
  <entry search="ApplicationIF" replace="RealApplication"
    src="globalDir" file="App/wncs_app.pr" />
  <entry search="MiddleWareIF" replace="RealMiddleWare"
    src="svnSDL" file="wncs_mw.pr" target="verbose" />
</Mapping>
```

## Tool Integration


Organizer rw communication.sdt <@rns2.agvs>
File Edit View Generate Tools Bookmarks ConTraST Tools MicaZ Workshop Help
/mnt/public/Projekte/SPaS
/mnt/public/Projekte/SPaS
MacIF
MAC
emptyProc
AppIF
Application
emptyProc
App2MacSignals
communication
app : Application
mac : MAC
Generate PR
Parse PR
» Generate «
Generate Code (NS-SDL)
Generate Code (with MSC)
Generate Code (with SEnF)
Generate Code (Imote2)
All Imote2 (PR+Contrast+make+upload)
Run Component
» Compile «
Compile Code
Make clean
Create documentation
Generate a PR file of the selected item and subitems





## Conclusion

**Problem**

Exchange of components (blocks, processes etc.) in SDL specifications

**Our Solution**

Concept of **SDL modules** with tool support

- Based on existing SDL language features
- Effective reuse of SDL components
- Effective adaptation of SDL systems to different scenarios and platforms
- SPaSs: standalone prototype tool
  - Tool demonstration during afternoon break
  - `http://vs.cs.uni-kl.de/activities/spass/`

**Questions?**