Ports primitifs et composites :
Une description formelle (exécutable) des propriétés de cohérence et complétude

Guy Tremblay, Nicolas Desnos, Christelle Urtado, Sylvain Vauttier, Marianne Huchard

Mai 2007

Table des matières

---

1 Introduction

Ce document décrit de façon formelle les notions présentées dans [1] : ports primitifs, composites, composants et assemblages, ainsi que certaines de leurs propriétés et relations, e.g., cohérence et complétude. Cette description formelle a pu être *vérifiée et compilée*, ce qui a donc permis, entre autres, de vérifier la cohérence des différentes définitions au niveau des déclarations de types. De plus, cette description a aussi pu être *exécutée* : les exemples prsentés à la section 6 ont donc été validés en les compilant et exécutant.

2 Outils et notations utilisés

Le présent document a été produit à l'aide de l'outil Letos [2] — *Lightweight Execution Tool for Operational Semantics*. L'intérêt de cet outil est double. Ainsi, à partir d'un document unique, il peut être utilisé de deux façons:

- Pour générer un document LaTeX, dans le style typique des sémantiques opérationnelles [7, 3, 4, 8].
- Pour générer *du code exécutable*, exprimé dans le langage fonctionnel (pur) Miranda [6]. En plus de pouvoir *animer* la spécification [5], ceci permet aussi de pouvoir effectuer des vérifications de type assurant que l'ensemble des définitions sont cohérentes.

Les éléments clés de la notation Letos, tels qu'ils apparaissent dans leur forme LaTeX, sont présentés à l'Annexe A.
Dans ce qui suit, la convention généralement utilisée pour distinguer les types des constructeurs ou fonctions est la suivante :

- Un identificateur en *italiques* débutant par une majuscule dénote un nom de type, par exemple, *Operation*, *Signature*, $Interface$.
- Un identificateur en police `teletype` débutant par une majuscule dénote un nom de constructeur (variante d'un type algébrique), par exemple, `Interface`, `Assembly`.
- Un identificateur en police `teletype` débutant par une minuscule dénote un nom de fonction, par exemple, `interfaces`, `operations`.

3 Définition des principaux concepts

3.1 Définition des types

Les types qui suivent décrivent les principales notions et concepts relatifs aux ports.

$$
\begin{array}{lcl}
Signature & \equiv & ([Type],\ Type); \\
Operation & \equiv & (OpName,\ Signature); \\
Interface & \equiv & \texttt{Interface}\ InterfaceName\ InterfaceKind\{Operation\}; \\
InterfaceKind & \equiv & \texttt{Provided} \mid \texttt{Required}; \\
Port & \equiv & \texttt{PrimPort}\ PortName\{Interface\} \mid \\
 & & \texttt{CompPort}\ PortName\{Port\}; \\
Component & \equiv & \texttt{Component}\{Interface\}\{Interface\}\{Port\}\{Port\}; \\
Connection & \equiv & (PortName,\ PortName); \\
Assembly & \equiv & \texttt{Assembly}\{Component\}\{Connection\}\{InterfaceName\};
\end{array}
$$

3.2 Attributs des entités des différents types

3.2.1 Attributs d'une interface

• Attribut pour obtenir le nom d'une interface :

$$
\begin{array}{lcl}
\texttt{name} & :: & Interface \rightarrow InterfaceName; \\
\texttt{name(Interface n k ops)} & = & \texttt{n};
\end{array}
$$

• Attribut pour obtenir la sorte d'une interface :

$$
\begin{array}{lcl}
\texttt{kind} & :: & Interface \rightarrow InterfaceKind; \\
\texttt{kind(Interface n k ops)} & = & \texttt{k};
\end{array}
$$

• Attribut pour obtenir les opérations définies dans une interface :

$$
\begin{array}{lcl}
\texttt{operations} & :: & Interface \rightarrow \{Operation\}; \\
\texttt{operations(Interface n k ops)} & = & \texttt{ops};
\end{array}
$$

3.2.2 Attributs d'un port

• Attribut pour obtenir le nom d'un port :

$$\begin{array}{lcl} \texttt{name} & :: & Port \rightarrow PortName; \\ \texttt{name(PrimPort pn intfs)} & = & \mathsf{pn}; \\ \texttt{name(CompPort pn ports)} & = & \mathsf{pn}; \end{array}$$

• Attribut pour obtenir les interfaces d'un port primitif :

$$\begin{array}{lcl} \texttt{interfaces} & :: & Port \rightarrow \{Interface\}; \\ \texttt{interfaces(PrimPort pn intfs)} & = & \mathsf{intfs}; \\ \texttt{interfaces(CompPort pn ports)} & = & \{\}; \end{array}$$

• Attribut pour obtenir les ports formant un port composite :

$$\begin{array}{lcl} \mathsf{ports} & :: & Port \rightarrow \{Port\}; \\ \mathsf{ports}(\texttt{CompPort pn prts}) & = & \mathsf{prts}; \\ \mathsf{ports}(\texttt{PrimPort pn intfs}) & = & \{\}; \end{array}$$

3.2.3 Attributs d'un composant

• Attribut pour obtenir les diverses interfaces d'un composant :

$$\begin{array}{lcl} \mathsf{providedInterfaces} & :: & Component \rightarrow \{Interface\}; \\ \mathsf{providedInterfaces}(\texttt{Component provs reqs prims comps}) & = & \mathsf{provs}; \\ & & \\ \mathsf{requiredInterfaces} & :: & Component \rightarrow \{Interface\}; \\ \mathsf{requiredInterfaces}(\texttt{Component provs reqs prims comps}) & = & \mathsf{reqs}; \\ & & \\ \mathsf{allInterfaces} & :: & Component \rightarrow \{Interface\}; \\ \mathsf{allInterfaces}(\texttt{Component provs reqs prims comps}) & = & \mathsf{provs} \cup \mathsf{reqs}; \end{array}$$

• Attributs pour obtenir les ports primitifs ou composites d'un composant :

$$\begin{array}{lcl} \mathsf{primitives} & :: & Component \rightarrow \{Port\}; \\ \mathsf{primitives}(\texttt{Component provs reqs prims comps}) & = & \mathsf{prims}; \\ & & \\ \mathsf{composites} & :: & Component \rightarrow \{Port\}; \\ \mathsf{composites}(\texttt{Component provs reqs prims comps}) & = & \mathsf{comps}; \end{array}$$

3.2.4 Attributs d'un assemblage

• Attributs pour obtenir les composants, les connections ou les objectifs fonctionnels d'un assemblage :

$$\begin{array}{lcl} \mathsf{components} & :: & Assembly \rightarrow \{Component\}; \\ \mathsf{components}(\texttt{Assembly comps conns intfs}) & = & \mathsf{comps}; \\ & & \\ \mathsf{connections} & :: & Assembly \rightarrow \{Connection\}; \\ \mathsf{connections}(\texttt{Assembly comps conns intfs}) & = & \mathsf{conns}; \\ & & \\ \mathsf{functObjectives} & :: & Assembly \rightarrow \{InterfaceName\}; \\ \mathsf{functObjectives}(\texttt{Assembly comps conns intfs}) & = & \mathsf{intfs}; \end{array}$$

• Prédicat pour déterminer si un port contient une interface identifée comme étant un des objectifs fonctionnels d'un assemblage :

$$\begin{array}{lcl} \mathsf{containsFunctObjective} & :: & Port \rightarrow Assembly \rightarrow \mathsf{bool}; \\ \mathsf{containsFunctObjective}\ \rho\ \ A & = & \exists\{\texttt{name intf} \in \mathsf{functObjectives}\ A \mid \mathsf{intf} \in \texttt{interfaces}\ \rho\ \}; \end{array}$$

3.3 Prédicats décrivant les conditions pour que diverses constructions soient valides

3.3.1 Conditions pour qu'un composant soit valide

```
valid     :: Component→bool;
valid C   = allProvided provs∧allRequired reqs∧allPrimPortValid∧allCompPortValid
            where
            allPrimPortValid  = ∀{isPrimitive ρ ∧interfaces ρ  ⊆ allInterfaces C | ρ ∈primitives C};
            allCompPortValid = ∀{valid γ  C | γ ∈composites C};
            provs             = providedInterfaces C;
            reqs              = requiredInterfaces C;
            ;
```

• Prédicats auxiliaires sur des interfaces :

```
allProvided       :: {Interface}→bool;
allProvided intfs  = ∀{kind intf = Provided | intf∈intfs};

allRequired       :: {Interface}→bool;
allRequired intfs  = ∀{kind intf = Required | intf∈intfs};
```

• Prédicats auxiliaires sur des ports :

```
isPrimitive                      :: Port→bool;
isPrimitive(PrimPort pn intfs)    = True;
isPrimitive p                     = False;

isComposite                      :: Port→bool;
isComposite(CompPort pn intfs)    = True;
isComposite p                     = False;
```

3.3.2 Conditions pour qu'un composite soit valide

```
valid       :: Port→Component→bool;
valid γ  C  = allPrimPortsValid∧allCompPortsValid,  if isComposite γ
              where
              allPrimPortsValid  = ∀{∀{interfaces ρ  ⊆ allInterfaces C | ρ ∈primPorts* p} | p∈ports γ };
              allCompPortsValid = ∀{[?] | p∈ports γ };
              ;
valid γ  C  = False,  otherwise;
```

Remarque : le dernière condition contient l'expression [?]. Il s'agit simplement d'une façon d'indiquer une valeur *à compléter ultérieurement*. En d'autres mots, la condition n'est spécifiée que de façon partielle et incomplète.

3.4 Types auxiliaires

```
Name           ≡ string;
Type           ≡ Name;
OpName         ≡ Name;
InterfaceName  ≡ Name;
PortName       ≡ Name;
```

4 Fonctions et relations auxiliaires sur les ports

4.1 Fonctions pour obtenir l'ensemble des ports associés directement ou indirectement à un port

• Fonction pour obtenir l'ensemble des ports primitifs contenus directement ou indirectement dans un port

```
primPorts*     :: Port→{Port};
primPorts* γ    = ⋃{primPorts* p | p∈ports γ },  if isComposite γ ;
primPorts* ρ    = {ρ },  if isPrimitive ρ ;
```

• Fonction pour obtenir l'ensemble des ports composites contenus directement ou indirectement dans un port :

$$\begin{array}{lcl}
\texttt{compPorts}^* & :: & Port \rightarrow \{Port\}; \\
\texttt{compPorts}^*\ \gamma & = & \bigcup\{\{\gamma'\} \cup (\texttt{compPorts}^*\ \gamma') \mid \gamma' \in \text{ports } \gamma \ \wedge\ \text{isComposite } \gamma' \},\ \ \textbf{if } \text{isComposite } \gamma\ ; \\
\texttt{compPorts}^*\ \rho & = & \{\},\ \ \textbf{otherwise};
\end{array}$$

4.2 Fonction pour déterminer si deux ports sont non reliés

$$\texttt{unrelated}(\gamma, \gamma') \ = \ \gamma \neq \gamma' \wedge \gamma \ \notin\ \texttt{compPorts}^*\ \gamma' \wedge \gamma' \ \notin\ \texttt{compPorts}^*\ \gamma\ ;$$

4.3 Fonction pour déterminer l'ensemble des ports primitifs partagés avec d'autres ports (composites) par un port composite

$$\texttt{shared}_C(\gamma) \ = \ \{\rho \mid \rho \in \texttt{primPorts}^*\ \gamma \ \wedge \exists\{\texttt{unrelated}(\gamma, \gamma') \mid \gamma' \in \text{composites } C\}\};$$

4.4 Fonctions auxiliaires pour déterminer si un port (primitif ou composite) est connecté relativement à un assemblage

$$\begin{array}{lcl}
\widehat{\rho_A} & = & \text{connectedIn } \rho \text{ (connections } A),\ \ \textbf{if } \text{isPrimitive } \rho \\
 & & \textbf{where} \\
 & & \text{connectedIn } \rho \ \text{ conns} = \exists\{\texttt{name}\ \rho \ \in \{\rho_1,\ \rho_2\} \mid (\rho_1,\ \rho_2) \in \text{conns}\}; \\
 & & ; \\
\widehat{\gamma_A} & = & \text{False},\ \ \textbf{otherwise}; \\
 & & \\
\widehat{\widehat{\gamma_A}} & = & \forall\{\widehat{\rho_A} \mid \rho \in \texttt{primPorts}^*\ \gamma\},\ \ \textbf{if } \text{isComposite } \gamma\ ; \\
\widehat{\widehat{\rho_A}} & = & \text{False},\ \ \textbf{otherwise};
\end{array}$$

5 Prédicats pour déterminer si un port ou un assemblage est cohérent

5.1 Cohérence d'un port

$$\begin{array}{lcl}
\texttt{coherent}_{C,A}(\gamma) & = & \text{allPrimsConnected} \vee \text{noPrimsConnected} \vee \text{partiallyConnCoherent},\ \ \textbf{if } \text{isComposite } \gamma \\
 & & \textbf{where} \\
 & & \text{allPrimsConnected} \ = \forall\{\widehat{\rho_A} \mid \rho \in \texttt{primPorts}^*\ \gamma\}; \\
 & & \text{noPrimsConnected} \ = \forall\{\neg(\widehat{\rho_A}) \mid \rho \in \texttt{primPorts}^*\ \gamma\}; \\
 & & \text{partiallyConnCoherent} = \text{p1} \wedge \text{p2} \\
 & & \qquad \textbf{where} \\
 & & \qquad \text{p1} = \forall\{\widehat{\rho_A} \ \Rightarrow\ \text{someUnrelated } \rho\ \ \gamma\ \ C\ A \mid \rho \in \texttt{shared}_C(\gamma)\}; \\
 & & \qquad \text{p2} = \forall\{\neg(\widehat{\rho_A}) \mid \rho \in \texttt{primPorts}^*\ \gamma\ \ \setminus\ \texttt{shared}_C(\gamma)\}; \\
 & & \qquad ; \\
 & & ; \\
\texttt{coherent}_{C,A}(\rho) & = & \text{False},\ \ \textbf{otherwise};
\end{array}$$

$$\text{someUnrelated } \rho\ \ \gamma\ \ C\ A \ = \ \exists\{\rho \ \in\ \texttt{primPorts}^*\ \gamma' \mid \gamma' \in \text{composites } C \wedge\ \texttt{unrelated}(\gamma, \gamma') \wedge\ \widehat{\widehat{\gamma'_A}}\};$$

5.2 Cohérence d'un composant

$$\texttt{coherent}_A(C) \ = \ \forall\{\texttt{coherent}_{C,A}(\gamma) \mid \gamma \in \text{composites } C\};$$

5.3 Complétude d'un assemblage

```
complete     ::  Assembly→bool;
complete A   =   allFunctPrimConnected∧allCompCoherent
                 where
                 allFunctPrimConnected = ∀{ρ̂_A | ρ ∈primitivePorts A ∧ containsFunctObjective ρ A}
                                         where
                                         primitivePorts A = ⋃{primitives comp | comp∈components A};
                                         ;
                 allCompCoherent       = ∀{coherent_A(comp) | comp∈components A};
                 ;
```

6 Quelques exemples

6.1 Exemple simple

```
absOp   =  ("abs", (["num"], "num"));
addOp   =  ("add", (["num", "num"], "num"));
incOp   =  ("inc", (["num"], "num"));
mulOp   =  ("mul", (["num", "num"], "num"));
printOp =  ("print", (["num"], "void"));
rndOp   =  ("rand", ([], "num"));

i1      =  Interface"i1"Provided{incOp};
i2      =  Interface"i2"Provided{absOp, addOp, mulOp};
i3      =  Interface"i3"Required{printOp};
i4      =  Interface"i4"Required{rndOp};

p0      =  PrimPort"p0"{i1, i2};
p1      =  PrimPort"p1"{i1, i3, i4};
p2      =  PrimPort"p2"{i1, i3};
p3      =  PrimPort"p3"{i3};
p4      =  PrimPort"p4"{i1, i3, i4};

g1      =  CompPort"g1"{p0, p1};
g2      =  CompPort"g2"{g1, p3, p4};

pp1     =  primPorts* g1;
pp2     =  primPorts* g2;

cp1     =  compPorts* g1;
cp2     =  compPorts* g2;
```

6.2 Exemple tiré du papier

6.2.1 Les interfaces

```
iQuestionC       = Interface"QuestionC"Required{};
iResponseC       = Interface"ResponseC"Provided{};
iDialogueC       = Interface"DialogueC"Required{};
iMoneyC          = Interface"MoneyC"Provided{};
iControlMB       = Interface"ControlMB"Provided{};
iTransactionMB   = Interface"TransactionMB"Provided{};
iStatisticsMB    = Interface"StatisticsMB"Provided{};
iTransBankMB     = Interface"TransBankMB"Required{};
iQuestionATM     = Interface"QuestionATM"Provided{};
iResponseATM     = Interface"ResponseATM"Required{};
iDialogueATM     = Interface"DialogueATM"Provided{};
iMoneyATM        = Interface"MoneyATM"Required{};
iControlATM      = Interface"ControlATM"Required{};
iTransactionATM  = Interface"TransactionATM"Required{};
iStockATM        = Interface"StockATM"Required{};
iOrderATM        = Interface"OrderATM"Provided{};
iStatisticsCB    = Interface"StatisticsCB"Required{};
iTransBankCB     = Interface"TransBankCB"Provided{};
```

6.2.2 Les ports primitifs

```
ppUserInfoC          = PrimPort"User_InfoC"{iQuestionC, iResponseC};
ppMoneyDialogueC     = PrimPort"Money_DialogueC"{iDialogueC, iMoneyC};
ppMoneyTransactionMB = PrimPort"Money_TransactionMB"{iControlMB, iTransactionMB};
ppRequestDataMB      = PrimPort"Request_DataMB"{iStatisticsMB, iTransBankMB};
ppUserInfoATM        = PrimPort"User_InfoATM"{iQuestionATM, iResponseATM};
ppMoneyDialogueATM   = PrimPort"Money_DialogueATM"{iDialogueATM, iMoneyATM};
ppMoneyTransactionATM= PrimPort"Money_TransactionATM"{iControlATM, iTransactionATM};
ppStockManagementATM = PrimPort"Stock_ManagementATM"{iStockATM, iOrderATM};
ppProvideDataCB      = PrimPort"Provide_DataCB"{iStatisticsCB, iTransBankCB};
```

6.2.3 Les ports composites

```
cpManageWithdrawMB = CompPort"Manage_WithdrawMB"{ppMoneyTransactionMB, ppRequestDataMB};
cpMoneyWithdrawATM = CompPort"Money_WithdrawATM"{ppMoneyDialogueATM, ppMoneyTransactionATM};
```

6.2.4 Les composants

```
cmpClient     = Component{iResponseC, iMoneyC}{iQuestionC, iDialogueC}{ppUserInfoC, ppMoneyDialogueC}{};
cmpMemberBank = Component{iControlMB, iTransactionMB, iStatisticsMB}{iTransBankMB}{ppMoneyTransactionMB, ppReques
cmpATM        = Component{iQuestionATM, iDialogueATM, iOrderATM}{iResponseATM, iMoneyATM, iControlATM, iTransac
cmpCentralBank= Component{iTransBankCB}{iStatisticsCB}{ppProvideDataCB}{};
```

6.2.5 L'assemblage décrit dans le papier

• La description de l'assemblage :

```
aFigure2 = Assembly comps conns functObjectives
           where
           comps          = {cmpClient, cmpMemberBank, cmpATM, cmpCentralBank};
           conns          = {c1, c2, c3}
                            where
                            c1 = ("Money_DialogueC", "Money_DialogueATM");
                            c2 = ("Money_TransactionMB", "Money_TransactionATM");
                            c3 = ("Request_DataMB", "Provide_DataCB");
                            ;
           functObjectives = {"DialogueC", "MoneyC"};
           ;
```

```
aAutre = Assembly comps conns functObjectives
         where
         comps          = {cmpClient, cmpMemberBank, cmpATM, cmpCentralBank};
         conns          = {c2, c3}
                          where
                          c2 = ("Money_TransactionMB", "Money_TransactionATM");
                          c3 = ("Request_DataMB", "Provide_DataCB");
                          ;
         functObjectives = {"DialogueC", "MoneyC"};
         ;
```

• Propriété de complétude de l'assemblage :

```
test1 = complete aFigure2;
```

6.3 D'autres assemblages, semblables, mais pas tout à fait, à celui décrit dans le papier

```
aAutre1 = Assembly comps conns functObjectives
          where
          comps          = {cmpClient, cmpMemberBank, cmpATM, cmpCentralBank};
          conns          = {c1, c3}
                           where
                           c1 = ("Money_DialogueC", "Money_DialogueATM");
                           c3 = ("Request_DataMB", "Provide_DataCB");
                           ;
          functObjectives = {"DialogueC", "MoneyC"};
          ;

aAutre2 = Assembly comps conns functObjectives
          where
          comps          = {cmpClient, cmpMemberBank, cmpATM, cmpCentralBank};
          conns          = {c1, c2}
                           where
                           c1 = ("Money_DialogueC", "Money_DialogueATM");
                           c2 = ("Money_TransactionMB", "Money_TransactionATM");
                           ;
          functObjectives = {"DialogueC", "MoneyC"};
          ;

aAutre3 = Assembly comps conns functObjectives
          where
          comps          = {cmpClient, cmpMemberBank, cmpATM, cmpCentralBank};
          conns          = {c1, c2, c3, c4}
                           where
                           c1 = ("Money_DialogueC", "Money_DialogueATM");
                           c2 = ("Money_TransactionMB", "Money_TransactionATM");
                           c3 = ("Request_DataMB", "Provide_DataCB");
                           c4 = ("User_InfoATM", "User_InfoC");
                           ;
          functObjectives = {"DialogueC", "MoneyC"};
          ;
```

• Propriétés de ces autres assemblages :

```
test2 = ¬ complete aAutre1;
test3 = ¬ complete aAutre2;
test4 = complete aAutre3;
```

A Présentation succincte de la notation Letos

Ce qui suit présente les éléments clés de la notation Letos, tels qu'ils apparaissent dans leur forme LaTeX :

- Définition d'un type *tuple* (enregistrement, produit cartésien) :

$$Signature == ([Type], Type);$$

  Une *Signature* est composée d'une *séquence* de *Type*s (les types des arguments) et d'un *Type* (le type du résultat).

- Définition d'un type (algébrique) *énuméré* :

$$InterfaceKind ::= \texttt{Provided} \mid \texttt{Required};$$

  Une valeur de type $InterfaceKind$ est un scalaire possédant l'une des deux valeurs indiquées.

- Définition d'un type (algébrique) avec différents constructeurs (variantes) :

$$\begin{aligned} Port ::= & \\ & \texttt{PrimPort}\ PortName\ \{Interface\} \mid \\ & \texttt{CompPort}\ PortName\ \{Port\}; \end{aligned}$$

  Un *Port* est soit un port primitif `PrimPort`, soit un port composite `CompPort`. Ces différentes sortes de port ont chacun des champs différents qui les caractérisent (attributs).

- Définition d'une fonction :

$$\begin{aligned} &\texttt{requiredInterfaces} :: Component \rightarrow\ Interface; \\ &\texttt{requiredInterfaces}(Component\ provs\ reqs\ prims\ comps) = reqs; \end{aligned}$$

Références

[1] N. Desnos, M. Huchard, C. Urtado, S. Vauttier, and G. Tremblay. Automated and unanticipated flexible component substitution. In *The 10th International ACM SIGSOFT Symposium on Component-Based Software Engineering (CBSE 2007)*, pages 33–48. Springer-Verlag, LNCS-4608, Medford, MA, July 2007.

[2] P. Hartel. LETOS—a lightweight execution tool for operational semantics. *Software—Practice and Experience*, 29(15):1379–1416, Dec. 1999.

[3] B. Meyer. *Introduction à la théorie des langages de programmation*. InterEditions, 1992. [QA76.7M4914].

[4] H. Nielson and F. Nielson. *Semantics with Applications: A Formal Introduction*. John Wiley & Sons, 1992.

[5] D. Turner. *Functional Progams as Executable Specifications*, volume Mathematical Logic and Programming Languages of *International Series in Computer Science*, pages 29–54. Prentice-Hall, 1984.

[6] D. Turner. An overview of miranda. *SIGPLAN Notices*, 21(12):158–166, Dec. 1986.

[7] D. Watt. *Programming language syntax and semantics*. Prentice Hall international series in computer science, 1991. [QA76.7 W38].

[8] G. Winskel. *The Formal Semantics of Programming Languages—An Introduction*. The MIT Press, 1993. Prêté par Lorne H.