- Tutoriel -

1- Objet

- L'objectif de se tutoriel est d'illustrer, sur un exemple simple, le fonctionnement de la chaîne d'outils (Objecteering, bMTL puis Eclipse5.7) pour la spécification et l'évaluation de contrats de Qualité de Service entre composants logiciels.
Ce travail se fonde sur UML2.0 pour la spécification des composants.

2- Exemple

- L'exemple, illustré par la Fig. 1, se compose de 5 composants : a, a1, a2, a3 et b. Le composant a est un composite dont les éléments constitutifs sont les composants a1, a2 et a3.
L'opération op1() fournie par le composant a est en fait déléguée au composant a1 qui la fournie concrètement. Pour réaliser op1(), le composant a1 requiert les opérations op2() et op3(), qui lui sont fournies respectivement par les composants a2 et a3 par assemblage.
Pour réaliser l'opération op2(), le composant a2 a besoin de l'opération op4(). Ce besoin est passé par délégation au composite a, puis comblé grâce à un assemblage entre les composants a et b.

- Il existe trois types de propriétés de Qualités de Service utilisés dans notre exemple : le délai maximum d'exécution d'une opération (timeOut T), la précision d'une opération[1] (precision E) et la mémoire consommée (memory M). La Fig. 1 représente les différents types de QoS associées aux diverses opérations.

- Le principe de base sur lequel est fondé ce travail est le suivant : le contrat principal réalisé par un composant est qu'il s'engage à fournir un ensemble d'opérations ssi son environnement lui fournit préalablement un ensemble d'opérations requises. Donc, toute opération fournie par un composant peut éventuellement dépendre d'une ou plusieurs opérations requises. La QoS de l'opération fournie dépend alors de la QoS des opérations requises.
Dans notre exemple, l'opération op1() réalisée par a1 dépend des opérations op2() et op3() réalisées respectivement par les composants a2 et a3.

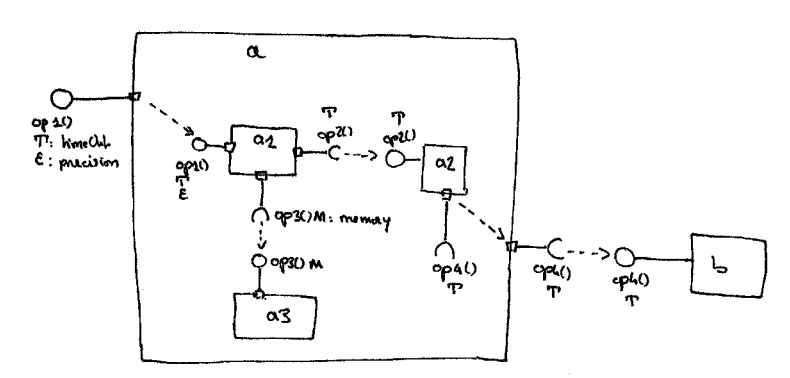


Fig. 1- Un exemple de diagrammes de composants

3- Métamodèle

- Les trois principaux diagrammes du métamodèle implanté (Fig. 2, Fig. 3 et Fig. 4) représente respectivement : 1) la spécification des opérations requises et fournies par un composant, 2) la structure interne d'un composite avec ses éléments internes et leurs connections, et 3) la QoS et les contrats de QoS réalisés par les composants.
La sémantique des deux premiers diagrammes relève d'UML2 et ne sera donc pas reprise dans ce document, à l'inverse de la sémantique du diagramme relatif à la QoS.

[1] Ceci fait référence au cas du GPS où une des QoS de l'opération 'getPosition' était la précision

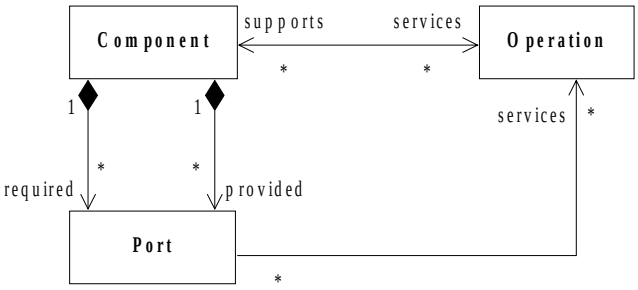


Fig. 2- Métamodèle : spécification des opérations

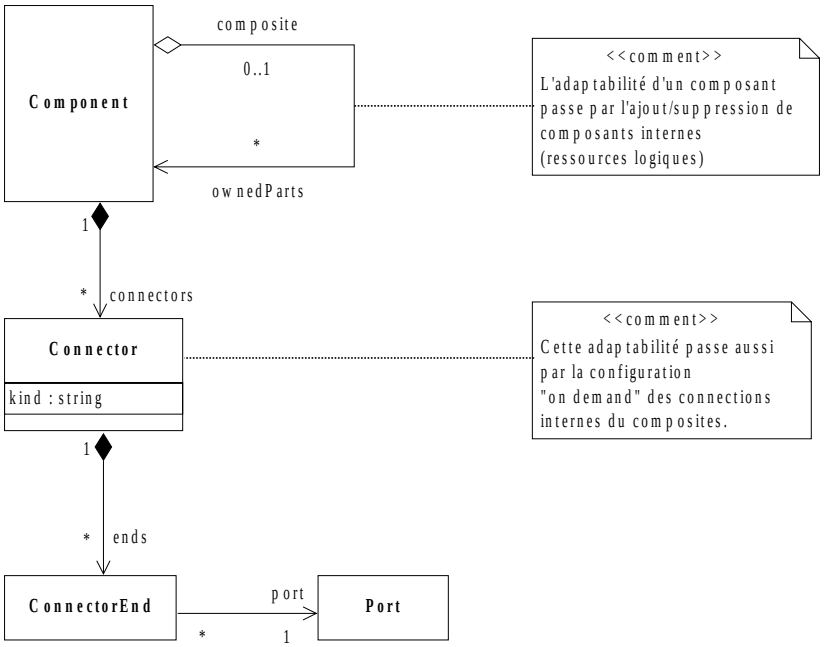


Fig. 3- Métamodèle : structures composites et connections

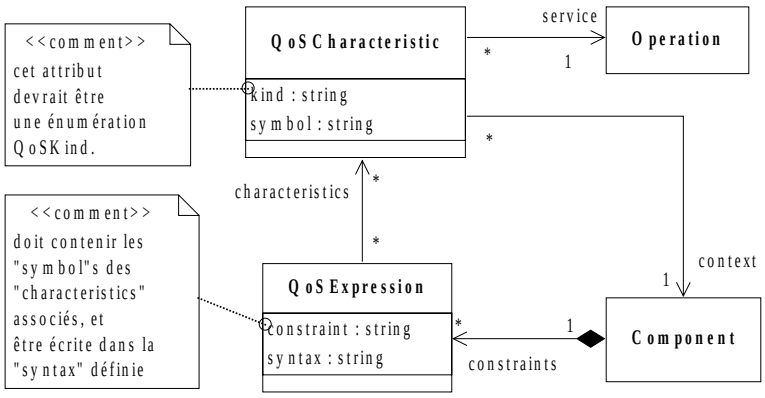
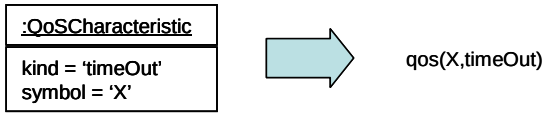


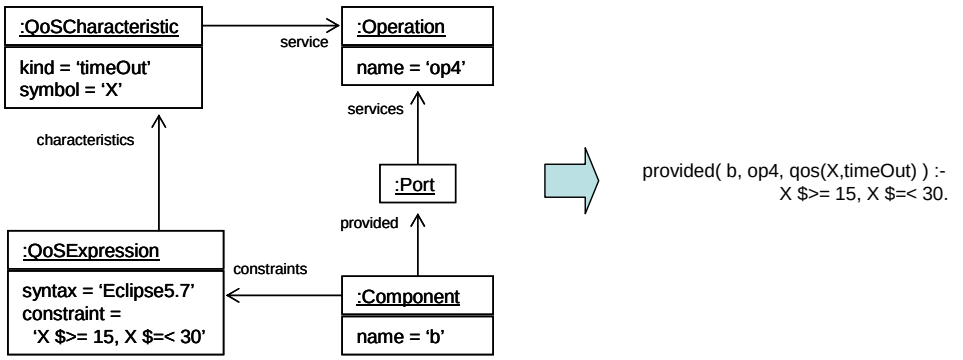
Fig. 4- Métamodèle : spécificatoin de la QoS

- Une QoSCharacteristic représente une propriété de QoS définie sur une opération particulière (association 'service'). Elle possède deux attributs : 1) 'kind' qui spécifie le type de la propriété (par exemple 'timeOut'), et 2) 'symbol' qui est une chaîne de caractère qui

identifie de manière unique cette propriété en tant que variable Prolog.
La transformation d'une instance de cette classe en un fait Prolog est illustrée par le schéma ci-dessous :



- Une QoSExpression représente un contrat de QoS réalisé par un composant. Elle réifie deux concepts : 1) la dépendance entre plusieurs propriétés de QoS spécifiées via l'association 'characteristics' ou 2) un niveau donné de QoS fournie si le nombre de propriétés de QoS est égal à un. L'attribut 'constraint' spécifie la contrainte dans une syntaxe particulière spécifiée grâce à l'attribut 'syntax' spécifie.
Dans l'exemple ci-dessous, le composant b fourni l'opération op4() laquelle est associée à une propriété de QoS de type 'timeOut' symbolisée par la variable 'X'. Ce composant déclare réaliser l'opération op4() avec une QoS de type 'timeOut' comprise entre 15 et 30 (15 < X <30).



Un extrait des 12 diagrammes Objecteering réalisant l'exemple complet est donné en annexe I.

4- Transformation

- La transformation (cf. annexe II) se décompose en deux étapes successives :

1. la transformation des QoSExpressions (dépendances et niveaux) dans une représentation Prolog ;
2. la transformation des Connectors (assemblage ou délégation).

- La première étape est décrite ci-dessus. La deuxième étape consiste à unifier les propriétés de QoS au niveau des connections entre les composants (assemblage ou délégation). En effet, prenons l'exemple suivant illustré par la Fig. 5 : soit op() une opération requise par un composant a et offerte par un composant b, connectés par un assemblage (non représentée sur la Fig. 5). Soit X le symbole de la propriété extra-fonctionnelle de type 'timeOut' associée à op().
Le composant a requiert que le niveau de la QoS de type 'timeOut' lors de toute exécution de op() soit inférieur à 20 (X $=< 20). Cette exigence, transformée en Prolog, s'écrit :

```
required( a, op, qos(X,timeOut) ) :-  X $=< 20.        [1]
```

De son côté, le composant b s'engage à réaliser op() avec une QoS de type 'timeOut' dont la valeur X sera toujours comprise entre 15 et 30. Ce contrat, transformée en Prolog, s'écrit :

```
provided( b, op, qos(X,timeOut) ) :-  X $>= 15, X $=< 30.        [2]
```

Il est évidement qu'à l'assemblage, il y a conjonction de ces deux contraintes. Or, au niveau de la transformation Prolog, le symbole X de la règle [1] n'est pas unifié avec le symbole X de la règle [2] : ce sont deux variables totalement libres, sans rapport aucun l'une avec l'autre. Il faut donc les unifier pour garantir la conjonction des deux contraintes :

```
required( a, op, qos(X,timeOut) ) :- provided( b, op, qos(X,timeOut) ).        [3]
```

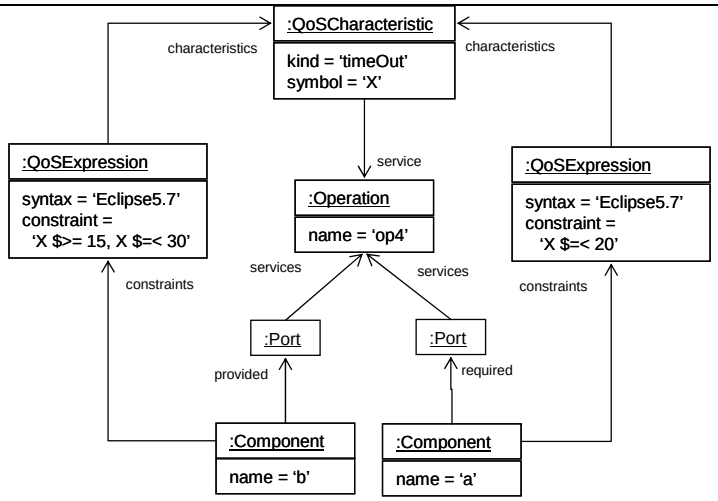


Fig. 5- Unification des propriétés de QoS

5- Utilisation

- Objecteering (équipé du module 'metamodelling' d'Erwan Drezen)permet donc d'instancier des modèles issus du métamodèle de QoS pour les composants que j'ai proposé. Grâce au module métamodelling, on peut exporter le métamodèle et les modèles sous forme de fichiers xml.

- La transformation (développée sous Eclipse) peut alors être appliquée pour générer automatiquement un fichier Prolog conforme à la syntaxe de Eclipse5.7 (à ne pas confondre avec la plateforme open-source Eclipse !)

- Voici deux exemples d'utilisation du fichier généré (cf. annexe III) avec Eclipse5.7 :

1. donner un encadrement de la valeur X de la propriété 'timeOut' associée à l'opération op1 fournie par le composant a :

```
:- provided( a, op1, qos( X, timeOut) ).
X = X{22.0 .. 25.0}
```

2. donner un encadrement de toutes les propriétés de QoS associées à l'opération op1 fournie par le composant a :

```
:- provided( a, op1, X ).
X = qos( T1{22.0 .. 25.0}, timeOut) ;
X = qos( medium, precision)
```

6- Configuration

- Liste des logiciels utilisés et leur numéro de version

- Objecteering 5.2.2
  - module Inria 'metamodelling' 1.0.n
- eclipse 3.0
  - plugin Inria MTL 0.0.5
- ECLiPSe 5.7_50

7- Références

- Jézéquel J.M., Defour O. and Plouzeau N. : '**An MDA approach to tame component based software development**' in J. S. de Boer, editor, *Post Proceedings of Formal Methods for Components and Objects (FMCO'03)*, number 3188 in LNCS. Springer Heidelberg, 2004.
- Defour O., Jézéquel J.M, and Plouzeau N. : '**Extra-functional contract support in components**' in *Proc. of International Symposium on Component-based Software Engineering (CBSE7)*, May 2004.
- Defour O., Jézéquel J.M., and Plouzeau N. : '**Applying CLP to predict extra-functional properties of component-based models**' in J. S. de Boer, editor, *Proceedings of Logic Programming: 20th International Conference, ICLP 2004*, number 3132 in LNCS. Springer Heidelberg, September 2004.

Annexe I- Diagrammes Objecteering

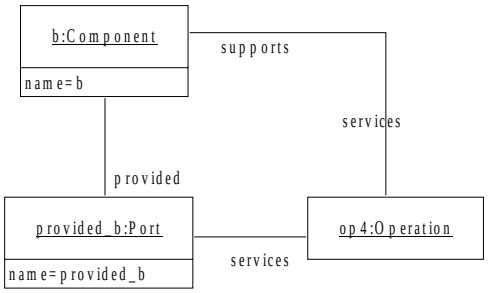


Fig. 6- Spécification des opérations du compsoant b

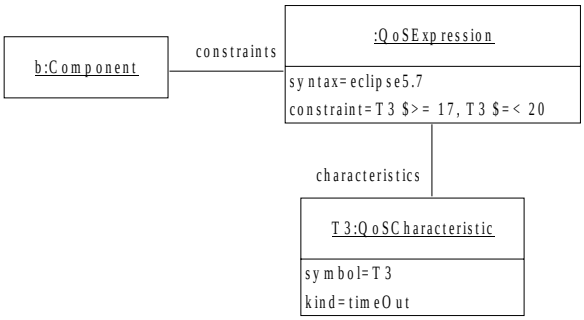


Fig. 7- Spécification d'un niveau de QoS

Fig. 8- Spécification d'une propriété de QoS

Annexe II- La transformation bMTL

```
library MyLib;

model source_model : RepositoryModel;

main() : Standard::Void {
        myClass : MyTransformationClass;
        genFile : Standard::String;

        'Model2Prolog v5.1'.toOut();
        myClass := new MyTransformationClass();
        myClass.connectSourceModel();
        genFile := '';
        genFile := myClass.qosDependency();
        genFile := genFile.concat( myClass.qosConnectors() );
        myClass.store(genFile);
}

class MyTransformationClass {

        /*
        *       Spécification du métamodèle et des modèles d'entrée et de sortie
        */
        connectSourceModel() {
                // variables locales
                mdrdriver : MDRDriver::MDRModelManager;
                metamodelFilename, inputFilename, outputFilename : Standard::String;
                set1, set2, set3 : Standard::Set;
                it1, it2 : Standard::Iterator;
                elt1, elt2 : source_model::Element;

                // spécification du métamodèle et des modèles d'entrée et de sortie
                metamodelFilename := 'Z:/clp/component/QoSComp_MOF.xml';
                inputFilename := 'Z:/clp/component/SmartExample.xml';
                outputFilename := 'Z:/clp/component/out.xml';

            // initialisation du driver MDR
```

```
		mdrdriver := new MDRDriver::MDRModelManager();
		mdrdriver.init();

		// instantiation du modèle d'entrée
		source_model := mdrdriver.getModelFromXMI (
			metamodelFilename,
			'QoSComp',	// name of the root package in the Uml1.4 metamodel
			'UML1.4_model',	// name of the new model in the repository
			inputFilename,
			outputFilename
		);
}

/*
*		Stockage des contraintes dans un fichier Prolog (Eclipse5.7)
*/
store( str : Standard::String ) {
		file : io::FileOutput;

		file := new io::FileOutput ();
		file.setName('Z:/clp/component/testGen');
		file.writeln(' % :-[\'//Z/clp/component/testGen\'].\n');
		file.writeln(' :- lib(ic).');
		file.writeln(str);
		file.close();
}

/*
*		parcourir tous les composants et traduire leurs
*		QoS dépendances en Prolog
*/
qosDependency() : Standard::String {
		set1, set2, set3 : Standard::Set;
		it1, it2, it3 : Standard::Iterator;
		comp : source_model::Component;
		qosexp : source_model::QoSExpression;
		qoschar : source_model::QoSCharacteristic;
		op : source_model::Operation;
		out, str0, str1, str2 : Standard::String;

		out := '';
		set1 := !source_model::Component!.allInstances();
		it1 := set1.getNewIterator();
		while it1.isOn() {
			comp := it1.item().oclAsType(!source_model::Component!);
			comp.name.toOut();
			set2 := comp.constraints;
			it2 := set2.getNewIterator();
			while it2.isOn() {
				str0 := '';
				qosexp := it2.item().oclAsType(!source_model::QoSExpresion!);
				set3 := qosexp.characteristics;
				it3 := set3.getNewIterator();
				while it3.isOn() {
					qoschar := it3.item().oclAsType(!source_model::QoSCharacteristic!);
					op := qoschar.service;
					if isProvided( op, comp ) {
						str1 := 'provided( ';
						str1 := str1.concat(comp.name.oclAsType(!Standard::String!));
						str1 := str1.concat(', ');
						str1 := str1.concat(op.name.oclAsType(!Standard::String!));
						str1 := str1.concat(', qos(');
						str1 := str1.concat(qoschar.symbol.oclAsType(!Standard::String!));
						str1 := str1.concat(',');
						str1 := str1.concat(qoschar.kind.oclAsType(!Standard::String!));
						str1 := str1.concat(') ) :-\n');
						str0 := str1.concat(str0);
					} else {
						str2 := '    required( ';
						str2 := str2.concat(comp.name.oclAsType(!Standard::String!));
						str2 := str2.concat(', ');
						str2 := str2.concat(op.name.oclAsType(!Standard::String!));
						str2 := str2.concat(', qos(');
						str2 := str2.concat(qoschar.symbol.oclAsType(!Standard::String!));
						str2 := str2.concat(',');
						str2 := str2.concat(qoschar.kind.oclAsType(!Standard::String!));
```

```
                                                str2 := str2.concat(') ), \n');
                                                str0 := str0.concat(str2);
                                        }
                                        it3.next();
                                }
                                str0 := str0.concat('    ');
                                str0 := str0.concat(qosexp.constraint.oclAsType(!Standard::String!));
                                str0 := str0.concat('.\n');

                                out := out.concat(str0);
                                it2.next();
                        }
                        it1.next();
                }
                return out;
}

isProvided( op : source_model::Operation;
                                comp : source_model::Component) : Standard::Boolean {
                res : Standard::Boolean;
                set : Standard::Set;
                it : Standard::Iterator;
                x : source_model::Component;

                res := false;
                set := op.supports;
                it := set.getNewIterator();
                while it.isOn() {
                        x := it.item().oclAsType(!source_model::Component!);
                        if ( comp = x ) {
                                res := true;
                        }
                        it.next();
                }
                return res;
}

/*
*               retourne les QoScharacteristic associés à une operation donnée
*/
getQoSChar( op : source_model::Operation ) : Standard::Set {
                set1, res : Standard::Set;
                it1 : Standard::Iterator;
                qoschar : source_model::QoSCharacteristic;
                x : source_model::Operation;
                flag : Standard::Boolean;

                res := new Standard::Set();
                flag := true;
                set1 := !source_model::QoSCharacteristic!.allInstances();
                it1 := set1.getNewIterator();
                while ( it1.isOn() and flag ) {
                        qoschar := it1.item().oclAsType(!source_model::QoSCharacteristic!);
                        x := qoschar.service;
                        if ( x = op ) {
                                res := res.including(qoschar);
                                flag = false;
                        }
                        it1.next();
                }
                return res;
}

/*
*               parcourir tous les connecteurs et traduire
*               cette connaissance en Prolog
*/
qosConnectors() : Standard::String {
                set1, set2, set3 : Standard::Set;
                it1, it2, it3 : Standard::Iterator;
                con : source_model::Connector;
                c1, c2 : source_model::ConnectorEnd;
                p1, p2 : source_model::Port;
                comp1, comp2 : source_model::Component;
                op : source_model::Operation;
                qoschar : source_model::QoSCharacteristic;
```

```
		out, str, qos : Standard::String;

		out := '';
		set1 := !source_model::Connector!.allInstances();
		it1 := set1.getNewIterator();
		while it1.isOn() {
			con := it1.item().oclAsType(!source_model::Connector!);
			set2 := con.ends;
			it2 := set2.getNewIterator();
			c1 := it2.item().oclAsType(!source_model::ConnectorEnd!); it2.next();
			c2 := it2.item().oclAsType(!source_model::ConnectorEnd!);
			p1 := c1.port.oclAsType(!source_model::Port!);
			p2 := c2.port.oclAsType(!source_model::Port!);
			comp1 := getComponent( p1 );
			comp2 := getComponent( p2 );

			set2 := p1.services;
			it2 := set2.getNewIterator();
			while it2.isOn() {
				op := it2.item().oclAsType(!source_model::Operation!);
				set3 := getQoSChar(op);
				it3 := set3.getNewIterator();
				while it3.isOn() {
					qoschar := it3.item().oclAsType(!source_model::QoSCharacteristic!);
					qos := 'qos(';
					qos := qos.concat(qoschar.symbol.oclAsType(!Standard::String!));
					qos := qos.concat(',');
					qos := qos.concat(qoschar.kind.oclAsType(!Standard::String!));
					qos := qos.concat(')');

					str := '';
					if isProvided( op, comp1) {
						str := str.concat('provided( ');
					} else {
						str := str.concat('required( ');
					}
					str := str.concat(comp1.name.oclAsType(!Standard::String!));
					str := str.concat(', ');
					str := str.concat(op.name.oclAsType(!Standard::String!));
					str := str.concat(', ');
					str := str.concat(qos);
					str := str.concat(') :-\n');
					if isProvided( op, comp2) {
						str := str.concat('    provided( ');
					} else {
						str := str.concat('    required( ');
					}
					str := str.concat(comp2.name.oclAsType(!Standard::String!));
					str := str.concat(', ');
					str := str.concat(op.name.oclAsType(!Standard::String!));
					str := str.concat(', ');
					str := str.concat(qos);
					str := str.concat(').\n');

					out := out.concat(str);

					it3.next();
				}
				it2.next();
			}
			it1.next();
		}
		return out;
}

/*
* renvoie le composant associé à un port donné
*/
getComponent( aPort : source_model::Port ) : source_model::Component {
		set1, set2 : Standard::Set;
		it1, it2 : Standard::Iterator;
		p : source_model::Port;
		comp, res : source_model::Component;
		flag : Standard::Boolean;

		flag := true;
```

```
                set1 := !source_model::Component!.allInstances();
                it1 := set1.getNewIterator();
                while ( it1.isOn() and flag ) {
                        comp := it1.item().oclAsType(!source_model::Component!);
                        set2 := comp.provided;
                        it2 := set2.getNewIterator();
                        while ( it2.isOn() and flag ) {
                                p := it2.item().oclAsType(!source_model::Port!);
                                if( p = aPort ) {
                                        res := comp;
                                        flag := false;
                                }
                                it2.next();
                        }
                        if( flag ) {
                                set2 := comp.required;
                                it2 := set2.getNewIterator();
                                while ( it2.isOn() and flag ) {
                                        p := it2.item().oclAsType(!source_model::Port!);
                                        if( p = aPort ) {
                                                res := comp;
                                                flag := false;
                                        }
                                        it2.next();
                                }
                        }
                        it1.next();
                }
                return res;
        }

}
```

Annexe III- La représentation Prolog de la QoS

```
% :-['//Z/clp/component/testGen'].

% librairie des contraintes sur les intervalles disponible dans Eclipse5.7
:- lib(ic).

% dépendances extra-fonctionnelles et niveaux de QoS
provided( b, op4, qos(T3,timeOut) ) :-
   T3 $>= 17, T3 $=< 20.

provided( a1, op1, qos(T1,timeOut) ) :-
   required( a1, op2, qos(T2,timeOut) ),
   T1 $= T2 + 3.

provided( a1, op1, qos(E1,precision) ) :-
   required( a1, op2, qos(T2,timeOut) ),
   required( a1, op3, qos(M,memory) ),
   rule( E1, T2, M).

provided( a2, op2, qos(T2,timeOut) ) :-
   required( a2, op4, qos(T3,timeOut) ),
   T2 $= T3 + 2.

provided( a3, op3, qos(M,memory) ) :-
   M = 128.

% unification des propriétés de QoS au niveau des délégations
provided( a, op1, qos(T1,timeOut)) :-
   provided( a1, op1, qos(T1,timeOut)).
```

```
provided( a, op1, qos(E1,precision)) :-
   provided( a1, op1, qos(E1,precision)).

required( a2, op4, qos(T3,timeOut)) :-
   required( a, op4, qos(T3,timeOut)).

% unification des propriétés de QoS au niveau des assemblages
required( a1, op3, qos(M,memory)) :-
   provided( a3, op3, qos(M,memory)).

required( a1, op2, qos(T2,timeOut)) :-
   provided( a2, op2, qos(T2,timeOut)).

required( a, op4, qos(T3,timeOut)) :-
   provided( b, op4, qos(T3,timeOut)).

% la règle qui lie trois variables E, T et M.
rule( low, T, M) :-
   ( T $>= 30, M $>= 0, M $< 500);
   ( M $>= 0, M $< 100, T $>= 20, T $< 30).
rule( medium, T, M) :-
    ( M $>= 0, M $< 100, T $>= 0, T $< 20);
          ( M $>= 100, M $< 500, T $>= 20, T $< 30);
          ( M $>= 500, T $>= 30).
 rule( high, T, M) :-
          (M $> 100, T $>= 0, T $< 20);
          (M $>= 500, T $>= 20, T $< 30 ).
```