



Rathaxes

Un DSL et son compilateur

David Pineau

`<david@lse.epitech.eu>`

5 juillet 2011

Qu'est-ce que c'est

Rathaxes est un compilateur et un langage dédié. L'objectif est de permettre la génération du code C d'un pilote de périphérique à partir d'une description algorithmique du périphérique. Le compilateur peut générer le code du pilote de périphérique en C pour n'importe quel système d'exploitation supporté.

Problématique du développement de pilote

1. Multiples compétences :

- ▶ Électronique : compréhension du matériel ;
- ▶ Développement noyau : connaissance de l'OS cible.

Problématique du développement de pilote

1. Multiples compétences :

- ▶ Électronique : compréhension du matériel ;
- ▶ Développement noyau : connaissance de l'OS cible.

2. Code critique ;

Problématique du développement de pilote

1. Multiples compétences :
 - ▶ Électronique : compréhension du matériel ;
 - ▶ Développement noyau : connaissance de l'OS cible.
2. Code critique ;
3. Réécriture permanente des pilotes pour chacun des OS cibles ;

Problématique du développement de pilote

1. Multiples compétences :
 - ▶ Électronique : compréhension du matériel ;
 - ▶ Développement noyau : connaissance de l'OS cible.
2. Code critique ;
3. Réécriture permanente des pilotes pour chacun des OS cibles ;
4. Évolution permanente de la problématique :

Problématique du développement de pilote

1. Multiples compétences :
 - ▶ Électronique : compréhension du matériel ;
 - ▶ Développement noyau : connaissance de l'OS cible.
2. Code critique ;
3. Réécriture permanente des pilotes pour chacun des OS cibles ;
4. Évolution permanente de la problématique :
 - ▶ Les systèmes évoluent ;

Problématique du développement de pilote

1. Multiples compétences :
 - ▶ Électronique : compréhension du matériel ;
 - ▶ Développement noyau : connaissance de l'OS cible.
2. Code critique ;
3. Réécriture permanente des pilotes pour chacun des OS cibles ;
4. Évolution permanente de la problématique :
 - ▶ Les systèmes évoluent ;
 - ▶ Le matériel évolue.

Les solutions

- ▶ Séparation des compétences ;

Les solutions

- ▶ Séparation des compétences ;
- ▶ Vérifications multiples ;

Les solutions

- ▶ Séparation des compétences ;
- ▶ Vérifications multiples ;
- ▶ Réutilisabilité ;

Les solutions

- ▶ Séparation des compétences ;
- ▶ Vérifications multiples ;
- ▶ Réutilisabilité ;
- ▶ Adaptabilité.

Ce qui est nécessaire

- ▶ Souplesse, évolutivité du compilateur ;

Ce qui est nécessaire

- ▶ Souplesse, évolutivité du compilateur ;
- ▶ Redondance ;

Ce qui est nécessaire

- ▶ Souplesse, évolutivité du compilateur ;
- ▶ Redondance ;
- ▶ Trois cas d'utilisation de Rathaxes :

Ce qui est nécessaire

- ▶ Souplesse, évolutivité du compilateur ;
- ▶ Redondance ;
- ▶ Trois cas d'utilisation de Rathaxes :
 - ▶ Définition des sémantiques communes associées aux sous-systèmes ;

Ce qui est nécessaire

- ▶ Souplesse, évolutivité du compilateur ;
- ▶ Redondance ;
- ▶ Trois cas d'utilisation de Rathaxes :
 - ▶ Définition des sémantiques communes associées aux sous-systèmes ;
 - ▶ Implémentation des sémantiques pour chacun des OS ;

Ce qui est nécessaire

- ▶ Souplesse, évolutivité du compilateur ;
- ▶ Redondance ;
- ▶ Trois cas d'utilisation de Rathaxes :
 - ▶ Définition des sémantiques communes associées aux sous-systèmes ;
 - ▶ Implémentation des sémantiques pour chacun des OS ;
 - ▶ Implémentation d'un pilote dans un langage dédié à l'aide des sémantiques implémentées ;

Ce qui est nécessaire

- ▶ Souplesse, évolutivité du compilateur ;
- ▶ Redondance ;
- ▶ Trois cas d'utilisation de Rathaxes :
 - ▶ Définition des sémantiques communes associées aux sous-systèmes ;
 - ▶ Implémentation des sémantiques pour chacun des OS ;
 - ▶ Implémentation d'un pilote dans un langage dédié à l'aide des sémantiques implémentées ;
- ▶ Un paradigme de programmation supportant un processus de génération de code complexe.

Un DSL en trois parties

Développeur Driver



Description du
périphérique :

- ▶ Registres ;
- ▶ Algorithmes.

Un DSL en trois parties

*Développeur
Driver*



*Développeur
Rathaxes*

Identification des
sémantiques communes à
tous les systèmes
d'exploitation afin d'en tirer
un modèle générique.

Description du
périphérique :

- ▶ Registres ;
- ▶ Algorithmes.

Un DSL en trois parties

Développeur Driver



Développeur Rathaxes

Identification des sémantiques communes à tous les systèmes d'exploitation afin d'en tirer un modèle générique.

Développeur Système

Code spécifique aux systèmes d'exploitation. Requiert des connaissances systèmes poussées.

Description du périphérique :

- ▶ Registres ;
- ▶ Algorithmes.

Un DSL en trois parties

Front-End

Fichier .rtx

Il contient une description :

- ▶ Physique du matériel (registres) ;
- ▶ Algorithmique du pilote ;
- ▶ Une configuration (sous-systèmes à utiliser, informations spécifiques à chaque système d'exploitation...).

Middle-End

Fichier .rti

Il contient des interfaces de sous-systèmes décrivant :

- ▶ Des Types ;
- ▶ Des Séquences ;
- ▶ Des variables de configuration.

Back-End

Fichier .blt

Il permet d'implémenter le code système-spécifique correspondant aux interfaces associées. On y retrouvera :

- ▶ Des templates de type ;
- ▶ Des templates de séquence.

Chaque morceau de code C instrumenté qui s'y trouve est appelé "placeHolder".

Comment faire un langage souple ?

Il existe plusieurs besoins en plus de celui d'avoir un compilateur et un langage évolutifs :

1. Implémenter facilement de nouvelles sémantiques ;

Comment faire un langage souple ?

Il existe plusieurs besoins en plus de celui d'avoir un compilateur et un langage évolutifs :

1. Implémenter facilement de nouvelles sémantiques ;
2. Spécifier le système pour lequel on implémente le code ;

Comment faire un langage souple ?

Il existe plusieurs besoins en plus de celui d'avoir un compilateur et un langage évolutifs :

1. Implémenter facilement de nouvelles sémantiques ;
2. Spécifier le système pour lequel on implémente le code ;
3. Générer facilement du C à partir de ces implémentations ;

Comment faire un langage souple ?

Il existe plusieurs besoins en plus de celui d'avoir un compilateur et un langage évolutifs :

1. Implémenter facilement de nouvelles sémantiques ;
2. Spécifier le système pour lequel on implémente le code ;
3. Générer facilement du C à partir de ces implémentations ;
4. Avoir un langage compréhensible.

Du langage souple aux templates

Ceci impliquait donc un langage instrumentant le C contenant :

1. Un moyen de sélectionner du code selon une configuration ;

Du langage souple aux templates

Ceci impliquait donc un langage instrumentant le C contenant :

1. Un moyen de sélectionner du code selon une configuration ;
2. Une association un élément = une sémantique ;

Du langage souple aux templates

Ceci impliquait donc un langage instrumentant le C contenant :

1. Un moyen de sélectionner du code selon une configuration ;
2. Une association un élément = une sémantique ;
3. Un moyen de manipuler cette surcouche au C dans le C.

Du langage souple aux templates

Se traduisant en un langage de templates contenant :

1. Un bloc “with” décrivant les conditions de sélection du code (configuration) ;
2. Un bloc “template” implémentant une sémantique décrite dans une interface ;
3. Une syntaxe “placeholder” permettant de manipuler des données du C instrumenté dans le C.

Comment résoudre un template ?

Un template ne peut pas :

1. Être indépendant de son contexte ;

Comment résoudre un template ?

Un template ne peut pas :

1. Être indépendant de son contexte ;
2. Générer du code brut ;

Comment résoudre un template ?

Un template ne peut pas :

1. Être indépendant de son contexte ;
2. Générer du code brut ;
3. Générer quelque chose d'incompatible avec le C.

Comment résoudre un template ?

Un template ne peut pas :

1. Être indépendant de son contexte ;
2. Générer du code brut ;
3. Générer quelque chose d'incompatible avec le C.

Il doit donc :

1. Se résoudre à l'aide de paramètres facilement manipulables ;

Comment résoudre un template ?

Un template ne peut pas :

1. Être indépendant de son contexte ;
2. Générer du code brut ;
3. Générer quelque chose d'incompatible avec le C.

Il doit donc :

1. Se résoudre à l'aide de paramètres facilement manipulables ;
2. Générer quelque chose de compatible avec le C ;

Comment résoudre un template ?

Un template ne peut pas :

1. Être indépendant de son contexte ;
2. Générer du code brut ;
3. Générer quelque chose d'incompatible avec le C.

Il doit donc :

1. Se résoudre à l'aide de paramètres facilement manipulables ;
2. Générer quelque chose de compatible avec le C ;
3. Utiliser une forme facilement manipulable.

Comment résoudre un template ?

Les solutions adoptées :

1. Mise en place d'un système de mapping de valeurs pour la résolution basé sur les paramètres du template ;
2. Génération d'un AST C compatible.

Accélérer le processus de génération

Qu'est-ce qui rend le processus de génération lent ?

1. Analyser/compiler un template a un coût ;

Accélérer le processus de génération

Qu'est-ce qui rend le processus de génération lent ?

1. Analyser/compiler un template a un coût ;
2. Rechercher un template précis est donc lent ;

Accélérer le processus de génération

Qu'est-ce qui rend le processus de génération lent ?

1. Analyser/compiler un template a un coût ;
2. Rechercher un template précis est donc lent ;
3. Le but de rathaxes est d'être une bibliothèque de templates et de drivers prêts à être générés.

Accélérer le processus de génération

Quelques idées pour améliorer les choses :

1. La compilation d'un template génère un AST et un script CodeWorker ;

Accélérer le processus de génération

Quelques idées pour améliorer les choses :

1. La compilation d'un template génère un AST et un script CodeWorker ;
2. Il faut maintenir un cache des templates compilés ;

Accélérer le processus de génération

Quelques idées pour améliorer les choses :

1. La compilation d'un template génère un AST et un script CodeWorker ;
2. Il faut maintenir un cache des templates compilés ;
3. Ce cache doit se limiter à des fichiers à charger avec le moins d'opérations possibles.

Accélérer le processus de génération

Le cache permet donc :

1. Accès rapide à un template en fonction de son prototype ;

Accélérer le processus de génération

Le cache permet donc :

1. Accès rapide à un template en fonction de son prototype ;
2. Chargement automatique du script et de l'arbre associés ;

Accélérer le processus de génération

Le cache permet donc :

1. Accès rapide à un template en fonction de son prototype ;
2. Chargement automatique du script et de l'arbre associés ;
3. Au travers de la fonction template générée, une résolution facile du template utilisé.

Apparition de nouvelles problématiques

Exemple de template tel que défini précédemment :

```
with LKM
values OS=Linux, version>2.6.30
{
  template init(Context)
  {
    struct module mod =
    {
      .open = ????,
    };
    int templated_init()
    {
      // Do something here
    }
    module_init(templated_init);
  }
}
```

Les manques des templates...

Plusieurs problématiques apparaissent :

1. Multiples insertions à un endroit précis par des sources différentes ;
2. Rendre une insertion optionnelle ;
3. Imaginer un modèle extensible résolvant ces problématiques.

...et donc la programmation aspectuelle

Les intérêts de la Programmation Orientée Aspect :

- ▶ Séparer des éléments de métiers différents au sein d'un code ;
- ▶ Centré sur l'idée de tissage de code statique ou dynamique.

...et donc la programmation aspectuelle

Les mots clefs du paradigme :

- ▶ Aspect ;
- ▶ Greffon (ou advice) ;
- ▶ Point de coupe (ou pointcut) ;
- ▶ Point de jonction (ou joinpoint).

Arrivée de l'aspectuel dans le DSL

Grâce à la Programmation Orientée Aspect, nous allons pouvoir :

1. Définir des points de coupe pour chaque élément nécessitant de multiples insertions ;

Arrivée de l'aspectuel dans le DSL

Grâce à la Programmation Orientée Aspect, nous allons pouvoir :

1. Définir des points de coupe pour chaque élément nécessitant de multiples insertions ;
2. Ajouter un comportement par défaut pour chacun des points de coupe ;

Arrivée de l'aspectuel dans le DSL

Grâce à la Programmation Orientée Aspect, nous allons pouvoir :

1. Définir des points de coupe pour chaque élément nécessitant de multiples insertions ;
2. Ajouter un comportement par défaut pour chacun des points de coupe ;
3. Permettre de définir des greffons (advices) dans les templates, pour spécialiser selon les situations :

Arrivée de l'aspectuel dans le DSL

Grâce à la Programmation Orientée Aspect, nous allons pouvoir :

1. Définir des points de coupe pour chaque élément nécessitant de multiples insertions ;
2. Ajouter un comportement par défaut pour chacun des points de coupe ;
3. Permettre de définir des greffons (advices) dans les templates, pour spécialiser selon les situations :
 - 3.1 Définition de fonction ;

Arrivée de l'aspectuel dans le DSL

Grâce à la Programmation Orientée Aspect, nous allons pouvoir :

1. Définir des points de coupe pour chaque élément nécessitant de multiples insertions ;
2. Ajouter un comportement par défaut pour chacun des points de coupe ;
3. Permettre de définir des greffons (advices) dans les templates, pour spécialiser selon les situations :
 - 3.1 Définition de fonction ;
 - 3.2 Définition de variables globales ;

Arrivée de l'aspectuel dans le DSL

Grâce à la Programmation Orientée Aspect, nous allons pouvoir :

1. Définir des points de coupe pour chaque élément nécessitant de multiples insertions ;
2. Ajouter un comportement par défaut pour chacun des points de coupe ;
3. Permettre de définir des greffons (advices) dans les templates, pour spécialiser selon les situations :
 - 3.1 Définition de fonction ;
 - 3.2 Définition de variables globales ;
 - 3.3 Appel de fonction ;

Arrivée de l'aspectuel dans le DSL

Grâce à la Programmation Orientée Aspect, nous allons pouvoir :

1. Définir des points de coupe pour chaque élément nécessitant de multiples insertions ;
2. Ajouter un comportement par défaut pour chacun des points de coupe ;
3. Permettre de définir des greffons (advices) dans les templates, pour spécialiser selon les situations :
 - 3.1 Définition de fonction ;
 - 3.2 Définition de variables globales ;
 - 3.3 Appel de fonction ;
 - 3.4 Attribution de pointeur sur fonction ;

Arrivée de l'aspectuel dans le DSL

Grâce à la Programmation Orientée Aspect, nous allons pouvoir :

1. Définir des points de coupe pour chaque élément nécessitant de multiples insertions ;
2. Ajouter un comportement par défaut pour chacun des points de coupe ;
3. Permettre de définir des greffons (advices) dans les templates, pour spécialiser selon les situations :
 - 3.1 Définition de fonction ;
 - 3.2 Définition de variables globales ;
 - 3.3 Appel de fonction ;
 - 3.4 Attribution de pointeur sur fonction ;
 - 3.5 etc...

Implémentation dans le langage

1. Ajout de *placeholders* point de coupe : les “pointcuts” ;

Implémentation dans le langage

1. Ajout de *placeholders* point de coupe : les “pointcuts” ;
2. Ajout d'un bloc “default” dans le point de coupe ;

Implémentation dans le langage

1. Ajout de *placeholders* point de coupe : les “pointcuts” ;
2. Ajout d'un bloc “default” dans le point de coupe ;
3. Changements du template :

Implémentation dans le langage

1. Ajout de *placeholders* point de coupe : les “pointcuts” ;
2. Ajout d'un bloc “default” dans le point de coupe ;
3. Changements du template :
 - 3.1 Déplacement du code dans des greffons : les “chunks” ;

Implémentation dans le langage

1. Ajout de *placeholders* point de coupe : les “pointcuts” ;
2. Ajout d'un bloc “default” dans le point de coupe ;
3. Changements du template :
 - 3.1 Déplacement du code dans des greffons : les “chunks” ;
 - 3.2 Ajout de greffons spécifiques pour cas particuliers (builtins).

Implémentation dans le cache

1. Recherche par template (ex : appel d'un template en Rathaxes) ;

Implémentation dans le cache

1. Recherche par template (ex : appel d'un template en Rathaxes) ;
2. Recherche par greffon (résolution d'un pointcut) ;

Implémentation dans le cache

1. Recherche par template (ex : appel d'un template en Rathaxes) ;
2. Recherche par greffon (résolution d'un pointcut) ;
3. Adaptabilité de la génération : génération de script et AST par configuration.

Pour résumer...

- ▶ Description des “aspects” du compilateur par des interfaces ;
- ▶ Implémentation des templates respectant ces interfaces ;
- ▶ Le langage utilisateur n'est qu'une suite de tissage de ces aspects.

Syntaxe d'une interface de sous-système

Fournit cinq types d'éléments :

- ▶ Types ;
- ▶ Séquences ;
- ▶ Variables ;
- ▶ Pointcuts ;
- ▶ Chunks.

Syntaxe d'une interface de sous-système

Quatre mots clefs pour identifier qui implémente quoi. Ces mot clefs s'appliquent aux séquences, types et variables.

- ▶ provided ;
- ▶ required ;
- ▶ optional ;
- ▶ auto.

Syntaxe d'une interface de sous-système

Deux mots clefs pour identifier qui implémente quoi pour la partie aspectuelle de Rathaxes.

- ▶ `provided ;`
- ▶ `use.`

Syntaxe d'une interface de sous-système

Tableau de croisement des mot clefs avec leur implication pour le Front-End et le Back-End :

	Front	Back	Generation
auto	Non	Oui	Oui
provided	Non	Oui	Si utilisé
required	Oui	Oui	Oui
optional	Possible	Oui si redéfini	Si utilisé

Exemple d'interface (.rti)

```

/*
 * This interface describes the basic needs for
 * any loadable kernel module
 */
interface LKM : Builtins
{
    builtin type      Device;

    provided pointcut include_dependencies;
    provided pointcut global_data_declaration;
    provided pointcut code_declaration;
    provided pointcut lkm_base_code_definition;

    provided sequence load()
    {
        provided pointcut lkm_init_fptrs;
        use      pointcut lkm_base_code_definition;
    }

    provided sequence unload()
    {
        provided pointcut unload_setup;
        use      pointcut lkm_base_code_definition;
    }
    required variable Builtins::string OS;
    required variable Builtins::serial version;
}

```

Exemple de template (.b1t)

```

with LKM
{
    ${pointcut include_dependencies};
    ${pointcut global_data_declaration};
    ${pointcut code_declaration};
    ${pointcut lkm_base_code_definition};
}
with LKM
values OS=Linux, version>=2.6.24
{
    template sequence load()
    {
        chunk lkm_base_code_definition
        {
            int modentry()
            { /* Init some stuff here */ }
            module_init(modentry);
        }
        chunk global_data_declaration
        {
            struct module myModule = {
                ${pointcut lkm_init_fptrs}
                default:
                    .module_open = NULL;
            },
        };
    }
}
}

```

Exemple d'implémentation de pilote (.rtx)

```
device myDriver
{
    // Here the driver's registers description
}

driver myDriver
{
    Userland::open(Context ctx) {
        log("Opening device...\n");
    }
    Userland::close(Context ctx) {
        log("Closing device...\n");
    }
}

configuration
{
    LKM::devices = myDriver;
    LKM::arch = x86;
    LKM::OS=Linux {
        LKM::version = 2.6.34;
    }
    LKM::OS=Windows7 {
    }
};
```

La différence avec un langage structuré

- ▶ Le code écrit n'a aucun rapport avec le code généré en termes de flux d'exécution ;

La différence avec un langage structuré

- ▶ Le code écrit n'a aucun rapport avec le code généré en termes de flux d'exécution ;
- ▶ Correspondance complexe entre déclaration rathaxes et expression C.

Du langage au compilateur

Trois parties dans le DSL :

- ▶ Indépendantes pour la compilation ;
- ▶ Liées pour la génération.

Du langage au compilateur

Trois parties dans le DSL :

- ▶ Indépendantes pour la compilation ;
- ▶ Liées pour la génération.

Le DSL est aussi un sur-ensemble du C :

- ▶ Gestion du langage C ;
- ▶ Instrumentalisation du C ;
- ▶ Interfacage pour la génération entre l'AST Rathaxes et l'AST du C.

Les composants

- ▶ CNorm ;

Les composants

- ▶ CNorm ;
- ▶ rtxParse : syntaxe du DSL ;

Les composants

- ▶ CNorm ;
- ▶ rtxParse : syntaxe du DSL ;
- ▶ RtxNode : normalisation des nodes de l'AST ;

Les composants

- ▶ CNorm ;
- ▶ rtxParse : syntaxe du DSL ;
- ▶ RtxNode : normalisation des nodes de l'AST ;
- ▶ rtxLink : cache des interfaces et templates enregistrés ;

Les composants

- ▶ CNorm ;
- ▶ rtxParse : syntaxe du DSL ;
- ▶ RtxNode : normalisation des nodes de l'AST ;
- ▶ rtxLink : cache des interfaces et templates enregistrés ;
- ▶ rtxTpl : résolution des templates ;

Les composants

- ▶ CNorm ;
- ▶ rtxParse : syntaxe du DSL ;
- ▶ RtxNode : normalisation des nodes de l'AST ;
- ▶ rtxLink : cache des interfaces et templates enregistrés ;
- ▶ rtxTpl : résolution des templates ;
- ▶ Tool : génération de fichiers annexes (.inf, Makefiles, etc...).

Compilation du Middle-End

- ▶ Analyse du code ;

Compilation du Middle-End

- ▶ Analyse du code ;
- ▶ Validation des dépendances ;

Compilation du Middle-End

- ▶ Analyse du code ;
- ▶ Validation des dépendances ;
- ▶ Validation de l'interface elle-même ;

Compilation du Middle-End

- ▶ Analyse du code ;
- ▶ Validation des dépendances ;
- ▶ Validation de l'interface elle-même ;
- ▶ Enregistrement dans le cache.

Compilation du Back-End

- ▶ Analyse du code C instrumenté ;

Compilation du Back-End

- ▶ Analyse du code C instrumenté ;
- ▶ Validation des prototypes de templates ;

Compilation du Back-End

- ▶ Analyse du code C instrumenté ;
- ▶ Validation des prototypes de templates ;
- ▶ Validation de la présence des éléments de programmation aspectuelle requis par l'interface ;

Compilation du Back-End

- ▶ Analyse du code C instrumenté ;
- ▶ Validation des prototypes de templates ;
- ▶ Validation de la présence des éléments de programmation aspectuelle requis par l'interface ;
- ▶ Extraction des PlaceHolders ;

Compilation du Back-End

- ▶ Analyse du code C instrumenté ;
- ▶ Validation des prototypes de templates ;
- ▶ Validation de la présence des éléments de programmation aspectuelle requis par l'interface ;
- ▶ Extraction des PlaceHolders ;
- ▶ Enregistrement dans le cache.

Compilation du Front-End

- ▶ Analyse du code ;

Compilation du Front-End

- ▶ Analyse du code ;
- ▶ Vérification des types utilisés à l'aide des interfaces ;

Compilation du Front-End

- ▶ Analyse du code ;
- ▶ Vérification des types utilisés à l'aide des interfaces ;
- ▶ Sélection des templates utilisés (appels, redéfinitions, etc...) depuis le cache ;

Compilation du Front-End

- ▶ Analyse du code ;
- ▶ Vérification des types utilisés à l'aide des interfaces ;
- ▶ Sélection des templates utilisés (appels, redéfinitions, etc...) depuis le cache ;
- ▶ Première étape de génération : LKM ;

Compilation du Front-End

- ▶ Analyse du code ;
- ▶ Vérification des types utilisés à l'aide des interfaces ;
- ▶ Sélection des templates utilisés (appels, redéfinitions, etc...) depuis le cache ;
- ▶ Première étape de génération : LKM ;
- ▶ Résolution des *placeholders* (templates/chunks/pointcuts) ;

Compilation du Front-End

- ▶ Analyse du code ;
- ▶ Vérification des types utilisés à l'aide des interfaces ;
- ▶ Sélection des templates utilisés (appels, redéfinitions, etc...) depuis le cache ;
- ▶ Première étape de génération : LKM ;
- ▶ Résolution des *placeholders* (templates/chunks/pointcuts) ;
- ▶ Assemblage et génération.

Merci

Questions ?