NEXEYA SYSTEMS

A NEXEYA Company

TANGO DeviceServers
Design & Implementation Guidelines

Sommaire

- Pourquoi ?
- Design
- Implémentation

NEXEYA SYSTEMS

A NEXEYA Company

Pourquoi ?

TANGO

NEXEYA SYSTEMS
A NEXEYA Company

Pourquoi ?

- **Collaboration entre instituts est obligatoire**
- **Difficultés à réutiliser certains développements**
  - Environnement trop spécifique
  - Configuration « codée en dur »
- **Améliorations du code**
  - Robustesse
  - Comportement (machine d'états…)
- **Difficultés pour un premier développement**
- **Fournir un cadre pour une collaboration plus efficace**

Pourquoi ?

- **Un guide des différentes règles de conception et d'implémentation d'un Device Server disponible sur le lien suivant:**

http://www-controle.synchrotron-soleil.fr:8001/docs/TangoGuidelines/TangoDesignGuidelines-GB4-3.pdf

- **Rédigé dans le cadre de la collaboration SOLEIL / MAX IV,**

- **Issu de l'expérience de:**
  - Soleil Synchrotron,
  - MAX IV,
  - L'ESRF,
  - Nexeya Systems.

Design

Design

- **Processus conseillé pour le développement d'un Device Tango**
  - Définir dans un cahier des charges les attributs, commandes, états du Device ainsi que son rôle
    - Permet de définir clairement et rapidement l'interface POGO
  - Ne pas hésiter à faire un diagramme de classe, des diagrammes de séquences quand les traitements sont un peu complexes (thread, plugins…)
  - Développer le Device
  - Effectuer une revue de code avec la checklist (ajouter des points de vérification si nécessaire)

Design

- **Réutilisabilité**
  - Toujours regarder si un device n'a pas été développé
  - Essayer d'être le plus réutilisable/extensible
  - Configurable et portable

- **Etats d'un device**
  - Définir clairement la machine d'état qui reflète les différents états possible, et les transitions associées.
  - L'état d'un device doit refléter l'état interne du système à n'importe quel moment.

Design

- **Définition de l'interface**
  - Commandes et attributs définies avec POGO
  - Toujours utiliser un attribut pour présenter une donnée produite par le device
  - Utiliser une commande pour une action (void-void)

- **Disponibilité du service**
  - Le temps de réponse / réactivité sont les indicateurs de performance d'un device.
  - Idéalement, l'implémentation du device doit assurer la disponibilité qu'importe la charge cliente externe ou la charge interne.

- **Deux solutions :**
  - Mécanisme de polling TANGO
  - Mécanisme de thread géré par le développeur

Design

- **Mécanisme de threading**
  - C++ : utilisation du thread yat4tango::DeviceTask
  - Utilise une file de messages dépilée par le thread
  - Se commande comme une classe « classique », par des méthodes en interface qui s'envoient en interne des messages
  - Hériter de yat4tango::DeviceTask
  - Définir les types de messages à gérer
  - Cf exemple AttributeSequenceWriterTask
  - Voir la doc YAT, YAT4TANGO (SourceForge)
  - Pour info, les librairies YAT et YAT4TANGO pour C++ offrent quantité d'outils (threading, plugins, mémoire partagée, strings…)

NEXEYA SYSTEMS

A NEXEYA Company

TANGO

Implémentation

Implémentation

- **La communauté TANGO est internationale, donc dans l'optique de partager les développements, l'anglais est préconisé pour :**
  - La définition des interfaces (attributs et commandes)
  - La documentation du device (aide en ligne pour l'utilisation des commandes et la description des attributs
  - Les commentaires insérés dans le code par le développeur
  - Les messages d'erreur
  - Le nom des variables et des commandes internes ajoutées par le développeur

Implémentation

- **Types**
  - Les types utilisés pour l'interface sont des types TANGO (non modifiable)
  - Le développeur choisit ses types pour son propre code, tant que ce n'est pas dépendant de la plateforme.

- **Code généré**
  - Le code généré automatiquement par POGO ne doit pas être modifié par le développeur
  - Le développeur doit inclure son propre code dans les zones « PROTECTED REGION »

Implémentation

- **Interface du device**
  - Les noms doivent être explicites et permettre de comprendre directement la nature de l'attribut, de la commande ou de la propriété
  - Ex: pour une alimentation, on aura

    outputCurrent (et non OC1)

    ActivateOutput1 (et non ActO1)

    PortNumber (et non prt)

- **Les recommandations sur le nommage sont :**
  - Nom composé d'au moins 2 caractères
  - Seulement des caractères alphanumériques (pas de _ ou -)
  - Démarre avec une minuscule(attributs), majuscule (commande, propriété)
  - Dans le cas d'un nom composé, chaque sous-mot doit commencer par une majuscule
  - N'utiliser aucun terme vague (ex: readValue)

- **Essayer de toujours nommer de la même façon le même genre d'information**
  - Ex : **intensity** = current = I = intens

Implémentation

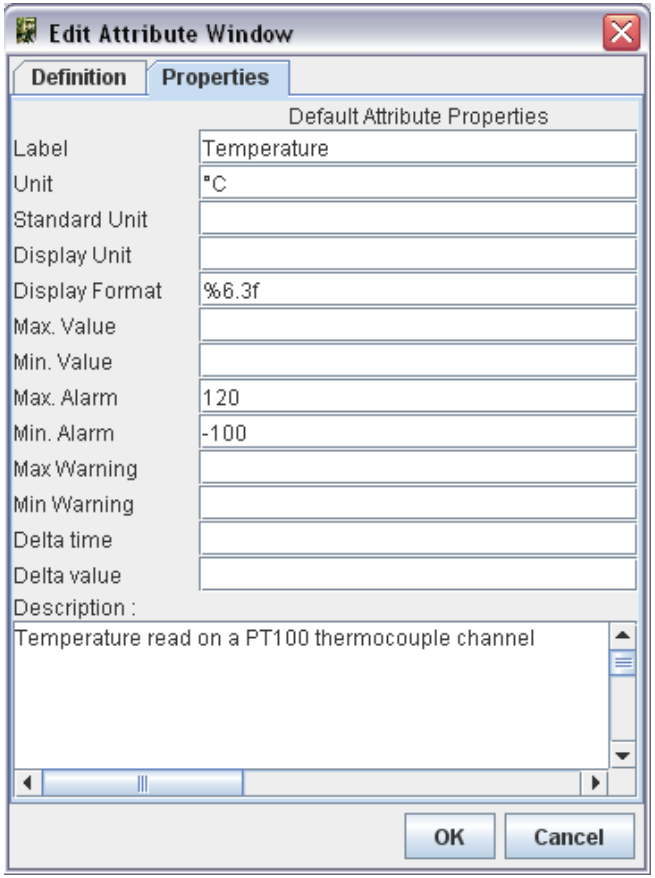
- **Choix du type de données**
  - Toujours utiliser un type de données en rapport avec l'information qu'il supporte
  - Ex : utiliser un ulong pour une information de type « numSamples », qui ne peut être négative

- **Choix du niveau d'interface**
  - Le choix entre Expert et Operator pour une interface doit être réfléchi
  - Les commandes de bases doivent toutes être accessibles pour l'Operator.
  - Les commandes pour un réglage fin de l'appareil seront réservées pour l'Expert. (métrologie, maintenance…)

Implémentation

Utilisation de POGO

- Toujours utiliser POGO pour créer ou modifier l'interface d'un device
- Chaque commande, attribut, propriété ou état d'un device doit être complètement documenté avec POGO.



Implémentation

- **Implémentation interne du device**
  - L'interface TANGO signifie qu'on peut l'utiliser dans un système de contrôle/commande.
  - Il faut penser le design interne comme n'importe quelle autre application et seulement ajouter l'interface TANGO par dessus.

- **Dans la pratique, il faut éviter de mixer le code généré avec POGO avec le code du développeur**

Implémentation

- **Etats d'un device**
  - ON, OFF, CLOSE, OPEN, INSERT, EXTRACT, MOVING, STANDBY, FAULT, INIT, RUNNING, ALARM, DISABLE, UNKNOWN

- **Il est recommandé, pour un matériel de type moteur, fente, monochromateur, et plus généralement pour tout équipement pouvant changer de position, de positionner l'état du device à « MOVING » lorsque l'équipement est en « mouvement » vers son point de consigne.**

- **Pour la méthode init_device, on recommande :**
  - si la phase d'initialisation est longue, il faut la threader
  - L'état INIT d'un device doit être réservé dans les phases de démarrages du device. L'état du device changera à la fin de l'initialisation.

Implémentation

- **Etats d'un device**

- **La sémantique recommandée pour les états FAULT, ALARM, UNKNOWN est la suivante:**
  - UNKNOWN (gris) : problème de communication avec l'équipement ou un sous-device qui empêche le device de savoir son vrai état.
  - FAULT (rouge) : problème qui empêche un fonctionnement normal (même pendant l'initialisation). Sortir de l'état FAULT est possible seulement en résolvant le problème et/ou en exécutant une commande Reset
  - ALARM (orange) : le device est fonctionnel mais un élément est « out of range » (attribut, butée d'un moteur, mauvais paramètres…)

Implémentation

- **Etats d'un device**
  - L'état « FAULT » ne dois pas tout interdire
- **L'exécution de la commande « Init » doit être réservée pour la réinitialisation du device (reconnexion hardware après reboot de l'équipement ou reconfiguration après un changement de valeur d'une propriété)**
- **Possibilité de créer sa propre machine d'état**
  - Surcharge des fonctions State et Status

Implémentation

- **Gestion des traces**

- **Toujours utiliser les streams**
  - DEBUG_STREAM : information développeur (trace de passage)
  - INFO_STREAM : information utilisateur (mesure, start/stop d'un processus)
  - WARN_STREAM : alerte hors erreur (fonctionnement dégradé par exemple)
  - ERROR_STREAM : erreur de fonctionnement
  - FATAL_STREAM : erreur fatale (arrêt du système)

- **Ajouter un attribut Log, tableau de string**
  - En C++, Yat, InnerAppender

Implémentation

- **Gestion des erreurs**
- **Toujours s'assurer :**
  - que toute exception est « *catchée* », complétée (TANGO permet cela) et propagée (utilisation de la méthode rethrow_exception),
  - que le code retour d'une fonction est toujours analysé,
  - que le *Status* du device est toujours en accord avec le *State*,
  - que les messages d'erreur sont compréhensibles pour « l'utilisateur moyen » et qu'ils sont complétés par des *logs* (de niveau *ERROR, utilisation de la macro error_stream*). Le *status* est l'indicateur qui va permettre à l'opérateur de comprendre d'où vient l'erreur.
- **Ignorer la situation idéale, cela n'est que trop peu rarement le cas.**
- **Imaginer les cas fréquents d'erreurs : cable non connecté, sous-device en FAULT…**

Implémentation

- Gestion des erreurs
- Utilisation de noms standardisés pour le champ reason des exceptions
- Afin de discriminer facilement les exceptions, il est recommandé d'utiliser une liste finie de types d'erreur pour le champ Reason, à indiquer en MAJUSCULE :

Nom standardisé pour les types d'erreur
OUT_OF_MEMORY
HARDWARE_FAILURE
SOFTWARE_FAILURE
HDB_FAILURE
DATA_OUT_OF_RANGE
COMMUNICATION_BROKEN
OPERATION_NOT_ALLOWED
DRIVER_FAILURE
UNKNOW_ERROR
CORBA_TIMEOUT
TANGO_CONNECTION_FAILED
TANGO_COMMUNICATION_ERROR
TANGO_WRONG_NAME_SYNTAX_ERROR
TANGO_NON_DB_DEVICE_ERROR
TANGO_WRONG_DATA_ERROR
TANGO_NON_SUPPORTED_FEATURE_ERROR
TANGO_ASYNC_CALL_ERROR
TANGO_ASYNC_REPLY_NOT_ARRIVED_ERROR
TANGO_EVENT_ERROR
TANGO_DEVICE_ERROR
CONFIGURATION_ERROR
DEPENDENCY_ERROR
NO_DEPENDENCY

Implémentation

- **Gestion des erreurs**
- **Toujours garder l'exception originale. Cela doit être la première chose visible dans le status.**
- **S'il y a une série d'exceptions, la logique veut que la première ait sans doute provoquée les autres. En résolvant la première exception, les autres peuvent disparaître.**

- **Gestion des erreurs dans la méthode init_device**
  - Aucune exception ne doit être propagée depuis cette méthode. Sinon le device se ferme. Le device doit toujours être vivant qu'importe son erreur.
  - Le code pour cette méthode doit contenir des blocs try/catch, qui garantiront qu'aucune exception n'est propagée dans ce contexte.
  - Si une exception est levée, le développeur doit mettre l'état à FAULT et mettre à jour le status en fonction de l'erreur. (*Le but est de comprendre facilement pourquoi le device a échoué dans son initialisation, tout en laissant la possibilité à l'opérateur de résoudre le ou les problèmes*)

Implémentation

- **Gestion des erreurs**
- **Cas particulier pour les attributs**
- **Il est recommandé de lever une exception pour toutes les valeurs invalides quelque soit leur type. Toutefois, il y a deux exceptions à cette règle : State et Status. Ces deux attributs doivent toujours retourner une valeur.**

- **Cas particuliers des propriétés dans init_device**
- **Si un exception TANGO est levée pendant la lecture des propriétés, le développeur doit systématiquement :**
  - Détecter l'erreur (catch)
  - La logger avec le level ERROR
  - Mettre le device en état FAULT
  - Mettre à jour le status en fonction de l'origine du problème

Questions ?