OpenRTM-aist (Java) - 機能 #2051

ComponentObserverConsumerの実装

2011/02/23 09:14 - n-ando

ステータス:	終了	**開始日:**	2011/02/23
優先度:	通常	**期日:**	
担当者:		**進捗率:**	100%
カテゴリ:		**予定工数:**	0.00時間
対象バージョン:			

説明

RTCの内部状態の変化をフックし通知するためのSDOサービスコンシューマ ComponentObserverConsumer を実装する。
以下、IDLファイルのドキュメントの抜粋である。

```
 * RTCの各種状態の更新を知らせるためのオブザーバーオブジェクトのため
 * のインターフェース。SDO Service として、対象となるRTC/SDOに対して
 * アタッチされ、RTC/SDO内の状態が変更された場合に、変更された状態の
 * 種類とヒントを同時に通知する。ツールなどで、ポーリングによらずRTC
 * の状態の変化を知りたい場合などに利用する。
 *
 * 想定している利用方法は以下のとおりである。
 *
 * -# SDO::get_configuration() により Configuration オブジェクトを取得
 * -# Configuration::add_service_profile() によりTool側の
 *      ComponentObserver を ServiceProfile により RTC に与える。
 *      ServiceProfile のメンバーは以下のように設定すること
 *    - id: UUID など一意なIDを設定する。削除時にも必要になるので、Tool
 *      側ではIDを保持しておかなければならない。
 *    - interface_type: 当該サービスのIFRのIDを文字列として指定。RTC側で
 *      はこの文字列により当該サービスオブジェクトを受け入れるか決定す
 *      るため指定は必須となる。
 *    - properties: RTC側のサービスの受け入れ側に通知するプロパティを設
 *      定する。このサービスでは、下記の heartbeat 関連のプロパティを
 *      指定する。
 *     - service: SDOService オブジェクトの参照を指定する。
 * -# RTC側で状態の変化があった場合に update_status() オペレーション
 *      が StatusKind および hint の文字列とともに呼び出される。Tool側
 *      では、StatusKind と hint に基づき RTC のある部分の状態が変化し
 *      たことを知り、必要な処理を行う。
 * -# 最終的にComponentObserverオブジェクトが不要になった場合には、
 *      Configuration::remove_service_profile() を id とともに呼び出し
 *      RTC から削除する。
 *
 * <pre>
 *
 *   [RTC]    [Configuration]           [Observer]    [Tool]
 *     |            |                        |            |
 *     |            | get_configuration()    |            |
 *     |<---------------------------------------------------|
 *     |            |                        |            |
 *     |            | add_service_profile(prof)           |
 *     |            |<-----------------------------------|
 *     |            |                        |            |
 *     |            | update_status(kind, hint)           |
 *     |----------------------------------->|            |
 *     |            | update_status(kind, hint)           |
 *     |----------------------------------->|            |
 *     |            |            :           |            |
 *     |            |                        |            |
 *     |            | remove_service_profile(id)          |
 *     |            |<-----------------------------------|
 *     |            |                        |            |
 *     |            |                        x            x
```

```
     *
     *  </pre>
     *
     *  なお、ServiceProfile::properties  に指定するプロパティとしては、
     *
     *  -  observed_status:  ALL  or  kind  of  status
     *  -  heartbeat.enable:  YES/NO
     *  -  heartbeat.interval:  x  [s]
     *
     *  がある。
     *
     *  -  observed_staus:  ALL  または状態の種類をカンマ区切りで指定
     *       監視する状態を指定する。指定可能な状態を表す文字列は、
     *       COMPONENT_PROFILE,  RTC_STATUS,  EC_STATUS,  PORT_PROFILE,
     *       CONFIGURATION  5種類である。監視したい対象をカンマで区切り複数指
     *       定することができる。また、すべての状態を監視する場合、ALL  を指定
     *       することができる。指定文字列は大文字、小文字を問わない。
     *
     *  -  heartbeat.interval:  秒単位で数値で指定
     *       ハートビートを送信する周期を秒単位で指定する。なお、指定した秒数
     *       でハートビートが必ず送信される保証はない。したがって、RTCが死ん
     *       だかどうかを確認するには、heartbeat.interval  数回分の時間を待つ
     *       必要がある。
     *
     *  -  heartbeat.enable:  YES  または  NOで指定
     *       Tool側では、状態に変化があるまで  RTC  が生存しているかどうか知る
     *       ことはできないため、突然RTCが死んだ場合には、これを知ることがで
     *       きない。そこで、HEART_BEAT  イベントを周期的にRTC側から送らせるこ
     *       とができる。ハートビートを有効にするか否かをこのオプションで指定
     *       する。
```

詳細はC++版の実装 r2050 を参照のこと。

関係しているリビジョン

リビジョン 534 - 2011/08/09 15:38 - fsi-katami

For implement of ComponentObserverConsumer. refs #2051

リビジョン 535 - 2011/08/09 16:03 - fsi-katami

Implemented ComponentObserverConsumer. refs #2051

リビジョン 536 - 2011/08/11 14:13 - fsi-katami

Implemented ComponentObserverConsumer. refs #2051

リビジョン 537 - 2011/08/11 15:54 - fsi-katami

Implemented ComponentObserverConsumer. refs #2051

リビジョン 538 - 2011/08/11 16:10 - fsi-katami

Implemented ComponentObserverConsumer. refs #2051

リビジョン 539 - 2011/08/11 18:04 - fsi-katami

Implemented ComponentObserverConsumer. refs #2051

リビジョン 540 - 2011/08/12 17:51 - fsi-katami

Implemented ComponentObserverConsumer. refs #2051

リビジョン 541 - 2011/08/13 15:59 - fsi-katami

Implemented ComponentObserverConsumer. refs #2051

リビジョン 542 - 2011/08/13 16:59 - fsi-katami

Implemented ComponentObserverConsumer. refs #2051

リビジョン 543 - 2011/08/14 10:07 - fsi-katami

Implemented ComponentObserverConsumer. refs #2051

リビジョン 544 - 2011/08/14 11:50 - fsi-katami

Implemented ComponentObserverConsumer. refs #2051

リビジョン 552 - 2011/09/06 17:49 - fsi-katami

Implemented ComponentObserverConsumer. refs #2051

履歴

#1 - 2011/08/12 17:56 - 匿名ユーザー

Tool の例

```
		try {
			conout.setObject(naming.resolve("ConsoleOut0.rtc"));
		} catch (NotFound e) {
			e.printStackTrace();
		} catch (CannotProceed e) {
			e.printStackTrace();
		} catch (InvalidName e) {
			e.printStackTrace();
		}
		try {

			org.omg.CORBA.Object obj = orb.resolve_initial_references("RootPOA");
			POA poa = POAHelper.narrow(obj);
			POAManager pman = poa.the_POAManager();
			pman.activate();
			Configuration conf = conout._ptr().get_configuration();

			UUID uuid = UUID.randomUUID();

			TestObserver to = new TestObserver();

			NVListHolder nvholder = new NVListHolder();
			nvholder.value = new NameValue[0];
			CORBA_SeqUtil.push_back(nvholder,
				NVUtil.newNVString("observed_status","ALL"));
			CORBA_SeqUtil.push_back(nvholder,
				NVUtil.newNVString("heartbeat.enable", "YES"));
			CORBA_SeqUtil.push_back(nvholder,
				NVUtil.newNVString("heartbeat.interval", "60"));
			byte[] id = poa.activate_object(to);
			org.omg.CORBA.Object ref = poa.id_to_reference(id);
			SDOService sdo = SDOServiceHelper.narrow(ref);
			ServiceProfile prof = new ServiceProfile(uuid.toString(),
                                                     ComponentObserverHelper.id(),
                                                     nvholder.value,
                                                     sdo);

			conf.add_service_profile(prof);
		}
		catch(Exception e){
			e.printStackTrace();
		}
```

Observer の例

```
public class TestObserver extends ComponentObserverPOA{
	public TestObserver() {
	}
	public void update_status (StatusKind status_kind, String hint){
		try{
			System.out.println( "--- update_status ---:"+status_kind.value()+":"+StatusKindHelper.type().member_name(status_kind.value())+"
		}
		catch(Exception e){
		}
	}
};
```

~
~
~
~
~

#2 - 2011/08/13 16:02 - 匿名ユーザー

fsi-katami は書きました:

> Tool の例
> [...]
>
> Observer の例
>
> [...]

#3 - 2011/08/14 10:56 - 匿名ユーザー

- ステータス を 新規 から 解決 に変更

- 進捗率 を 0 から 100 に変更

#4 - 2011/08/14 14:23 - 匿名ユーザー

- 進捗率 を 100 から 80 に変更

#5 - 2011/08/14 14:45 - 匿名ユーザー

sdo.service.provider.enabled_services の設定が反映されない。

#6 - 2011/09/06 18:18 - 匿名ユーザー

- 進捗率 を 80 から 100 に変更

#7 - 2012/02/04 04:13 - n-ando

- ステータス を 解決 から 終了 に変更