บันทึกวิจัย

การพัฒนา และออกแบบ

Information Grid Client API

Version 1.0

คำรณ อรุณเรื่อ และ นัยนา สหเวชชภัณฑ์

โครงการกริดสารสนเทศ

หน่วยปฏิบัติการวิจัยการจำลองขนาดใหญ่

ศูนย์เทคโนโลยีอิเล็กทรอนิกส์และคอมพิวเตอร์แห่งชาติ

กันยายน 2551

บทคัดย่อ

Information Grid Client API คือชุดคำสั่งสำหรับการพัฒนาแอพลิเคชั่นเพื่อติดต่อใช้บริการข้อมูลจากกริดสารสนเทศ (Information Grid) ที่ให้บริการข้อมูลในรูปแบบ stateful Service โดย statefule service นี้ได้รับการพัฒนาขึ้นบนพื้นฐานของ เทคโนโลยี Web Service Resource Framework (WSRF) ของ Globus Toolkit 4 (GT4) ทั้งนี้ชุดคำสั่งของ Information Grid Client API จะช่วยลดความซับซ้อนในการเรียกใช้บริการเว็บเซอร์วิส การจัดการรีซอร์ส การสืบค้นข้อมูล และการจัดการผลการสืบค้นข้อมูล โดยอิงรูปแบบชุดคำสั่งที่ผู้พัฒนาโปรแกรมทั่วไปคุ้นเคย ทั้งนี้ชุดคำสั่ง Information Grid Client API ในเวอร์ชั่นนี้จะสามารถใช้ได้กับแอพลิเคชั่นในแพลตฟอร์มภาษาจาวา

สารบัญ

1 บทนำ 6

2 การเขียนโปรแกรมเพื่อติดต่อกับ iBroker 8

2.1 อินเตอร์เฟสของ iBroker 8

2.2 ขั้นตอนการเขียนโปรแกรมเพื่อติดต่อกับ iBroker 8

3 ข้อกำหนดในการพัฒนา Information Grid Client API 11

3.1 รายละเอียดความต้องการของ Information Grid Client API 11

3.2 Use case diagram ของ Information Grid Client API 11

4.สถาปัตยกรรม และโครงสร้างของ Information Grid Client API 15

4.1 สถาปัตยกรรมของ Information Grid Client API 15

4.2 Package diagram 17

4.3 Class diagram 18

4.4 Sequence diagram 21

5.การเขียนโปรแกรมด้วย Information Grid Client API 24

5.1 การเขียนโปรแกรมเพื่อรับผลการสืบค้นข้อมูลในรูปแบบ “การส่งผลในคราวเดียวกัน” 24

5.2 การเขียนโปรแกรมเพื่อรับผลการสืบค้นข้อมูลในรูปแบบ “การทยอยการส่งผล” 25

5.3 การดักจับเหตุการณ์ (Events) ด้วย BrokerListener 27

6.บทสรุป 28

สารบัญภาพ

รูปที่ 2-1 ความสัมพันธ์ของขั้นตอนในการเขียนโปรแกรมเพื่อติดต่อ และเรียกใช้งาน iBroker....9

รูปที่ 2-2 ตัวอย่างการเขียนโปรแกรมเพื่อติดต่อ และเรียกใช้งาน iBroker....10

รูปที่ 3-1 Use case diagram สำหรับ Information Grid Client API....11

รูปที่ 4-1 สถาปัตยกรรมของ Information Grid Client API....15

รูปที่ 4-2 Package diagram ของ Information Grid Client API....17

รูปที่ 4-3 Class Diagram สำหรับแพคเกจ Broker Client....18

รูปที่ 4-4 Class Diagram สำหรับแพคเกจ Information Query....19

รูปที่ 4-5 Class Diagram สำหรับแพคเกจ QueryTool....20

รูปที่ 4-6 Sequence Diagram สำหรับการ Manage Endpoint....21

รูปที่ 4-7 Sequence Diagram สำหรับการติดต่อInformation Broker แบบ Immediate Command....22

รูปที่ 4-8 Sequence Diagram สำหรับการติดต่อInformation Broker แบบ Continuous Command....23

รูปที่ 5-1 ตัวอย่างโปรแกรม TestImmediateCommand.java....24

รูปที่ 5-2 ตัวอย่างโปรแกรม TestContinuousCommand.java....25

รูปที่ 5-3 ตัวอย่างโปรแกรม MyBroker.java....26

รูปที่ 5-3 BrokerListener interface....27

สารบัญตาราง

ตารางที่ 3-1 Use case “Instantiate iBroker via Immediate Command” 12

ตารางที่ 3-2 Use case “Instantiate iBroker via Continuous Command” 13

ตารางที่ 3-3 Use case “Manage Resource and Status” 13

ตารางที่ 3-4 Use case “Manage Event” 13

ตารางที่ 3-5 Use case “Define Query” 14

ตารางที่ 3-6 Use case “Execute Query” 14

ตารางที่ 3-7 Use case “Retreive Query” 14

1 บทนำ

ในการพัฒนาแอพลิเคชั่นบนพื้นฐานของกริดสารสนเทศ [] ผู้พัฒนาแอพลิเคชั่นจะต้องเขียนโปรแกรมเพื่อติดต่อกับ Information Broker (iBroker) ซึ่งเป็น stateful service ที่ได้รับการพัฒนาขึ้นบนพื้นฐานของ Globus Toolkit 4 (GT4) ทั้งนี้ ในการเขียนโปรแกรมเพื่อติดต่อกับ stateful service นั้น จะมีความแตกต่างจาก stateless service ที่ผู้พัฒนาฯทั่วไปคุ้นเคย ซึ่งสามารถจำแนกออกเป็น 2 ด้านหลักๆ คือ ด้านการกำหนดค่าอ้างอิงของ service และด้านการให้ได้มาซึ่งผลลัพธ์ที่ได้จากการประมวลผล โดยมีรายละเอียดดังนี้

<u>การกำหนดค่าอ้างอิงของ Service</u>

- **Stateless Service**: ค่าอ้างอิงของ stateless service หนึ่งๆ สามารถกำหนดได้โดยตรงผ่านทาง URL ที่อ้างอิงถึง WSDL ของ stateless service นั้นๆ
- **Stateful Service**: ค่าอ้างอิงของ stateful service หนึ่งๆ จะต้องกำหนดผ่านทาง WSDL ที่อ้างอิงถึง factory service ของ stateful service ตลอดจน resource ที่ factory service สร้างขึ้นมาต่อการเรียกใช้งาน stateful service หนึ่งๆ

<u>การให้ได้มาซึ่งผลลัพธ์ที่ได้จาก การประมวลผลของ Service</u>

- **Stateless Service**: การได้มาซึ่งผลลัพธ์จากการทำงานของ stateless service นั้น สามารถได้มาโดยตรงผ่านทางการเรียกใช้งานโอเปอร์ชั่นที่กำหนดไว้ใน stateless service ซึ่งทำหน้าที่ในการประมวลผล และส่งผลลัพธ์ที่ได้กลับไปยังแอพลิเคชั่นที่เรียกใช้งาน ทั้งนี้จะไม่มีการจัดเก็บผลลัพธ์เพื่อนำไปใช้ในการประมวลผลในครั้งถัดไป
- **Stateful Service**: การได้มาซึ่งผลลัพธ์จากการทำงานของ stateful service นั้น สามารถได้มาผ่านทางการเรียกใช้งาน 2 โอเปอร์ชั่นที่กำหนดไว้ใน stateful service โดยโอเปอร์เรชั่นหนึ่งทำหน้าที่ในการประมวลผล และนำผลลัพธ์ที่ได้ไปเก็บไว้ใน resource เพื่อนำไปใช้ในการประมวลผลครั้งต่อไป และอีกโอเปอร์เรชั่นหนึ่งทำหน้าที่ส่งผลลัพธ์ที่เก็บไว้ใน resource กลับไปยังแอพลิเคชั่นที่เรียกใช้งาน

จากข้างต้น จะเห็นว่าการเขียนโปรแกรมเพื่อติดต่อกับ stateful service มีความยุ่งยากและซับซ้อนกว่า stateless service และในการเขียนโปรแกรมเพื่อติดต่อกับ iBroker นั้น จะมีความยุ่งยากกว่า stateful service ดังที่ได้กล่าวไว้ข้างต้น เนื่องจาก iBroker สามารถทยอยการส่งผลลัพธ์ให้แอพลิเคชั่นที่มาติดต่อด้วย นอกเหนือจากการส่งผลลัพธ์ในคราวเดียวกัน ซึ่งในการให้ iBroker ทยอยการส่งผลลัพธ์ให้นั้น ผู้พัฒนาฯจะต้องเขียนโปรแกรมเพื่อตรวจสอบสถานะการทำงานของ iBroker ผ่านทางคุณสมบัติของ resource ก่อนที่จะทำการเรียกโอเปอร์เรชั่นให้ส่งผลลัพธ์ที่เก็บไว้ใน resource กลับมาให้ (ในกรณีที่ iBroker ได้ทำการบันทึกผลลัพธ์ใหม่เข้า resource) นอกจากนี้ ผู้พัฒนาฯยังต้องกำหนดคำสั่งการสืบค้นข้อมูล เพื่อส่งให้ iBroker ทำการสืบค้นข้อมูลให้ ซึ่งคำสั่งนี้ได้รับการออกแบบมาในรูปของ object ที่ผู้พัฒนาฯจะต้องทำความเข้าใจก่อนที่จะสามารถสร้างคำสั่งการสืบค้นข้อมูลได้

นอกเหนือจากการเขียนโปรแกรมเพื่อติดต่อกับ iBroker แล้ว การติดตั้งไลบรารี่ของ GT4 ยังมีรายละเอียดและขั้นตอนค่อนข้างเยอะ ซึ่งอาจทำให้ผู้พัฒนาฯดำเนินการ ติดตั้งโดยข้ามขั้นตอนใดขั้นตอนหนึ่ง ทั้งนี้จะส่งผลให้ผู้พัฒนาฯไม่สามารถทำการเขียนโปรแกรมเพื่อติดต่อกับ iBroker ได้

ดังนั้น หน่วยปฏิบัติการฯจึงมีแนวคิดในการพัฒนา Information Grid Client API (iGrid Client API) โดยมีวัตถุประสงค์หลักเพื่อซ่อนความซับซ้อนในการเขียนโปรแกรม และ การติดตั้ง GT4 ตลอดจนเพื่อสนับสนุนและส่งเสริมให้มีการพัฒนาแอพลิเคชั่นบนพื้นฐานของกริดสารสนเทศ บันทึกวิจัยฉบับนี้จึงได้นำเสนอถึงการออกแบบและพัฒนา iGrid Client API ซึ่งได้มุ่งเน้นไปยังการทำงานใน 3 ด้านหลักๆ ดังนี้

- การกำหนดค่าอ้างอิงของ iBroker

  iGrid Client API จะสนับสนุนให้ผู้พัฒนาฯสามารถทำการกำหนดค่าอ้างอิงของ iBroker ได้อย่างสะดวกขึ้น โดยจะซ่อนความซับซ้อนในการติดต่อกับ factory service และ resource ของ iBroker

- การสร้างคำสั่งการสืบค้นข้อมูล

  iGrid Client API จะสนับสนุนให้ผู้พัฒนาฯสามารถทำการกำหนดคำสั่งในการสืบค้นข้อมูลในรูปแบบของ Structured Query Language (SQL) ที่ผู้พัฒนาฯส่วนใหญ่คุ้นเคย แทนการใช้ผ่าน object ที่ออกแบบมาเฉพาะเพื่อใช้งานภายในกริดสารสนเทศ

- การรับผลการสืบค้นข้อมูลจาก iBroker

  iGrid Client API จะสนับสนุนให้ผู้พัฒนาฯสามารถรับผลสืบค้นข้อมูลโดยใช้กลไกของ Event-driven Programming ที่ผู้พัฒนาฯส่วนใหญ่คุ้นเคย แทนการตรวจสอบสถานะการทำงานของ iBroker ผ่านทางคุณ-สมบัติของ resource

ทั้งนี้ ในบันทึกวิจัยฉบับนี้ มีโครงสร้างดังนี้

- หัวข้อที่ 2 ได้ให้รายละเอียดถึงการเขียนโปรแกรมเพื่อติดต่อกับ iBroker
- หัวข้อที่ 3 ระบุข้อกำหนดในการพัฒนา iGrid Client API
- หัวข้อที่ 4 แสดงสถาปัตยกรรม และการทำงานของ iGrid Client API
- หัวข้อที่ 5 อธิบายถึงการเขียนโปรแกรมเพื่อติดต่อกับiBroker ผ่านทางI Grid Client API

2 การเขียนโปรแกรมเพื่อติดต่อกับ iBroker

2.1 อินเตอร์เฟสของ iBroker

ในการพัฒนาแอพลิเคชั่นบนพื้นฐานของกริดสารสนเทศนั้น ผู้พัฒนาฯจะต้องเขียนโปรแกรมเพื่อติดต่อกับ iBroker โดย iBroker มีอินเตอร์เฟสที่ประกอบด้วย 3 โอเปอร์เรชั่น ดังนี้

1. GetInformationResponse **getInformation** (GetInformation query) คือโอเปอเรชั่นที่ใช้สำหรับสั่งให้ iBroker ทำการสืบค้นและบูรณาการข้อมูล บนพื้นฐานของคำสั่งการสืบค้นข้อมูลที่กำหนดไว้ในพารามิเตอร์ query
2. GetResourcePropertyResponse **getResourceProperty** (QName name) คือโอเปอเรชั่นที่ใช้สำหรับเรียกดูสถานะการทำงานของ iBroker เช่น “สำเร็จ” “ล้มเหลว” หรือ “อยู่ในระหว่างการทำงาน” ตลอดจนใช้เรียกดูว่า iBroker ได้มีการบันทึกข้อมูลใหม่เข้า resource หรือไม่ ทั้งนี้ เพื่อที่ว่าแอพลิเคชั่นที่ติดต่อด้วยนั้น จะได้ตัดสินใจทำการร้องขอข้อมูลจาก iBroker
3. GetDataResultResponse **getDataResult** (GetDataResult data) คือโอเปอเรชั่นที่ใช้เพื่อร้องขอให้ iBroker ทำการส่งผลการสืบค้นข้อมูลให้ (หลังจากที่มีการเรียกใช้งานโอเปอเรชั่นgetInformation()) ทั้งนี้โอเปอร์เรชั่นนี้ควรถูกใช้งานควบคู่ไปกับโอเปอเรชั่น getResourceProperty() ซึ่งจำเป็นต้องเรียกคู่ของโอเปอเรชั่นเหล่านี้ซ้ำๆ จนกว่า iBroker จะดำเนินการสืบค้นและบูรณาการข้อมูลแล้วเสร็จ โดยมีสถานะการทำงานเป็น “สำเร็จ”

2.2 ขั้นตอนการเขียนโปรแกรมเพื่อติดต่อกับ iBroker

รูปที่ 2-2 ได้แสดงถึงตัวอย่างการเขียนโปรแกรมในการติดต่อกับ iBroker ซึ่งมีขั้นตอนทั้งหมด 5 ขั้นตอนหลัก โดยมีรายละเอียด ดังนี้ (โดยรูปที่ 2-1 แสดงความสัมพันธ์ของขั้นตอนในการเขียนโปรแกรมดังกล่าว)

1. กำหนดค่าอ้างอิงถึง iBroker ซึ่งมีขั้นตอนย่อยดังนี้
   - 1.1 กำหนดค่าอ้างอิง (endpoint reference) สำหรับ factory service ของ iBroker ซึ่งอยู่ในรูปของ InformationBrokerFactoryServiceAddressingLocator
   - 1.2 กำหนดให้ factory service ทำการสร้าง resource ต่อการเรียกใช้งาน iBroker หนึ่งๆ ซึ่งอยู่ในรูปของ CreateResourceResponse
   - 1.3 กำหนดค่าอ้างอิงสำหรับ iBroker ผ่านทาง factory service และ resource ที่ได้จากขั้นตอนที่ 1.1 และ 1.2 ซึ่งอยู่ในรูปของ InformationBrokerPortType

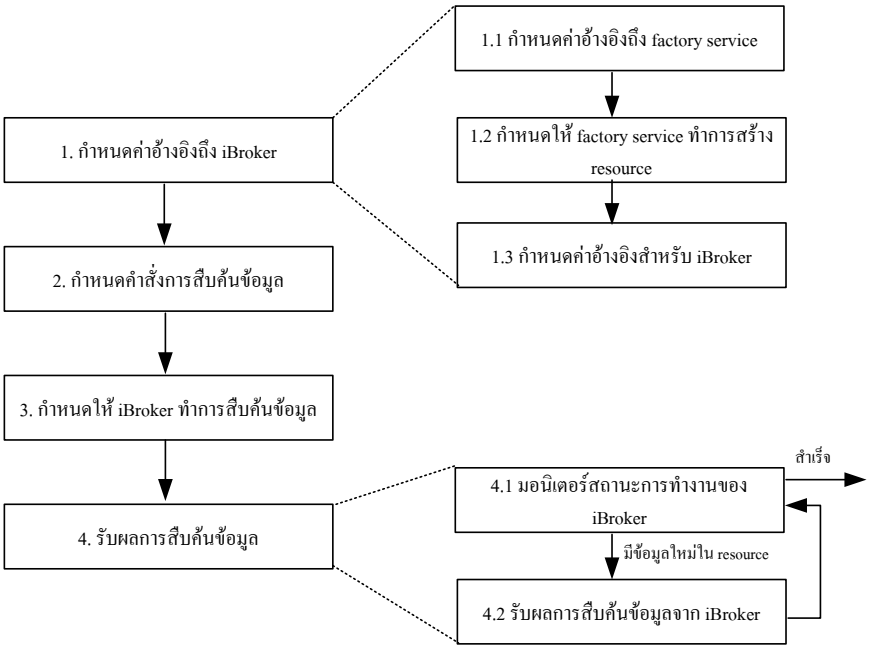
2. กำหนดคำสั่งการสืบค้นข้อมูล ในรูปของ GetInformation ซึ่งประกอบด้วย
   - MdlURI ซึ่งแสดงถึงรหัสมาตรฐานโครงสร้างข้อมูล
   - MdlQuery ซึ่งแสดงถึงชุดเงื่อนไขในการสืบค้นข้อมูล โดยแต่ละเงื่อนไขประกอบด้วยฟิลด์ข้อมูล โอเปอร์เรเตอร์ และค่าของฟิลด์ข้อมูลที่จะทำการสืบค้น

3. กำหนดให้ iBroker ทำการสืบค้นข้อมูล ผ่านทางการเรียกใช้งานโอเปอร์เรชั่น getInformation () ซึ่งมีพารามิเตอร์เป็นคำสั่งการสืบค้นข้อมูลที่ได้สร้างขึ้นในขั้นตอนที่ 2

4. รับผลการสืบค้นข้อมูลในรูปของข้อความที่มีเนื้อความเป็นเอกสาร XML ทั้งนี้ แอพลิเคชั่นที่มาติดต่อกับ iBroker สามารถระบุการรับผลได้ในรูปแบบของ “การรับผลในคราวเดียวกัน” หรือ “การทยอยการรับผล” โดยการได้มาซึ่งผลในคราวเดียวกันนี้ สามารถได้มาจากการดำเนินการในขั้นตอนที่ 3 ในขณะที่การรับผลแบบทยอย มีขั้นย่อยที่ต้องดำเนินการดังนี้

   4.1 มอนิเตอร์สถานะการทำงานของ iBroker ผ่านทางการเรียกใช้งานโอเปอร์เรชั่น getResourceProperty() ทั้งนี้เพื่อตรวจสอบถึงสถานะการทำงานของ iBroker ตลอดจนการบันทึกข้อมูลใหม่เข้า resource

   4.2 รับผลการสืบค้นข้อมูลจาก iBroker ผ่านทางการเรียกใช้งานโอเปอร์เรชั่น getDataResult () ซึ่ง iBroker จะส่งผลที่ได้ทำการบันทึกไว้ใน resource กลับคืนมาให้ ทั้งนี้ จะต้องดำเนินการในขั้นตอนที่ 4 และ 5 ควบคู่กันไปมาจนกว่า iBroker จะดำเนินการแล้วเสร็จ



รูปที่ 2-1 ความสัมพันธ์ของขั้นตอนในการเขียนโปรแกรมเพื่อติดต่อ และเรียกใช้งาน iBroker

```
EndpointReferenceType factoryEPR, instanceEPR;
InformationBrokerFactoryPortType factory;
InformationBrokerPortType service;
InformationBrokerFactoryServiceAddressingLocator fLocator = new InformationBrokerFactoryServiceAddressingLocator ();
factoryEPR = new EndpointReferenceType ();
factoryEPR.setAddress(new Address (<<endpoint URL>>));
factory = fLocator.getInformationBrokerFactoryPortTypePort(factoryEPR);
CreateResourceResponse createResponse = factory.createResource(new CreateResource ());
InstanceEPR = createResponse.getEndpointReference ();
service = fLocator.getInformationBrokerPortTypePort(instanceEPR);
GetInformation input = new GetInformation ();
input.setMdlURI("http://www.thai-research.net/mdl/complextype/researchproject");
MdlQuery mdlquery = new MdlQuery ();
MdlQueryParameter[] params = new MdlQueryParameter[1];
params[0] = new MdlQueryParameter();
params[0].setParameter("/*");
params[0].setParameter("Project/name");
params[0].setOperator(Operator.Li);
params[0].setValue(new MdlQueryParameterValue[] {new MdlQueryParameterValue("การวิจัย")});
mdlquery.setMdlQueryParameter(params);
input.setMdlQuery(mdlquery);
GetInformationResponse result = service.getInformation(input);

GetResourcePropertyResponse res = service.getResourceProperty(InformationBrokerQName.RP_LOCALMONITORINGSET);
LocalMonitoringSet lms = (LocalMonitoringSet) res.get_any()[0].getObjectValue(LocalMonitoringSet.class);
LocalMonitoringSetISources lmis = lms.getISources();

GetDataResultResponse gdr = service.getDataResult(new GetDataResult ());
org.igrid.isource.serviceType.GetInformationResponse[] ires = gdr.getGetInformationResponse();
gdr.getGetInformationResponse(0).getDataElement());
```

1.1 1.2 1.3 2 3 4.1 4.2

รูปที่ 2-2 ตัวอย่างการเขียนโปรแกรมเพื่อติดต่อ และเรียกใช้งาน iBroker

3 ข้อกำหนดในการพัฒนา Information Grid Client API

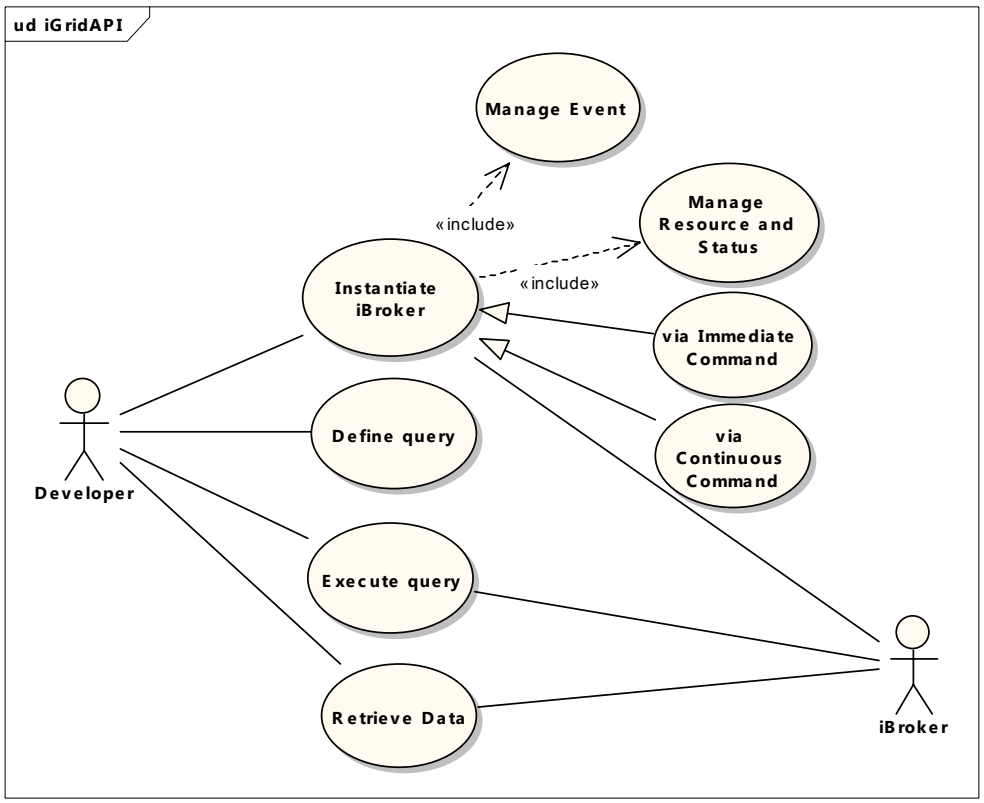
3.1 รายละเอียดความต้องการของ Information Grid Client API

จากหัวข้อที่ 2 จะเห็นว่าการเขียนโปรแกรมเพื่อติดต่อและเรียกใช้งาน iBroker นั้นมีความยุ่งยากและซับซ้อน ซึ่งสามารถจำแนกได้ออกเป็น 3 ส่วนหลักๆ คือ

1. ส่วนการกำหนดค่าอ้างอิงถึง iBroker
2. ส่วนการกำหนดคำสั่งการสืบค้นข้อมูล
3. ส่วนการรับผลการสืบค้นข้อมูล ที่อยู่ในรูปแบบของ “การทยอยการรับผล”

ดังนั้น จึงจำเป็นอย่างยิ่งที่หน่วยปฏิบัติการฯจะดำเนินการออกแบบ และพัฒนา Information Grid Client API (iGrid Client API) เพื่อซ่อนความยุ่งยากและซับซ้อนในการเขียนโปรแกรมเพื่อติดต่อกับ iBroker สำหรับ 3 ส่วนงานดังกล่าว

3.2 Use case diagram ของ Information Grid Client API



รูปที่ 3-1 Use case diagram สำหรับ Information Grid Client API

รูปที่ 3-1 แสดง Use case diagram ของ iGrid Client API ซึ่งสื่อให้เห็นถึงฟังก์ชั่นการทำงานที่ iGrid Client API จะต้องมีเพื่อสนับสนุนให้ผู้พัฒนาแอพลิเคชั่นสามารถทำการเขียนโปรแกรมเพื่อติดต่อกับ iBroker ได้สะดวกมากขึ้น ทั้งนี้ iGrid Client API ประกอบด้วย 7 use cases และ 2 actors ดังนี้

Use Case

1. Use case "Instantiate iBroker via Immediate Command" : จัดการกำหนดค่าอ้างอิงถึงตัวแทนของ iBroker ซึ่งมีการส่งผลการสืบค้นข้อมูล ในรูปแบบของ "การส่งผลในคราวเดียวกัน"
2. Use case "Instantiate iBroker via Continuous Command" : จัดการกำหนดค่าอ้างอิงถึงตัวแทนของ iBroker ซึ่งมีการส่งผลการสืบค้นข้อมูล ในรูปแบบของ "การทยอยการส่งผล"
3. Use case "Manage Event" : จัดการการสร้าง event ซึ่งบรรจุผลการสืบค้นที่ได้ไปยัง client
4. Use case "Manage Resource and Status": จัดการการตรวจสอบสถานะการทำงานของ iBroker และ resource เพื่อให้ทราบถึงการได้มาซึ่งข้อมูลที่ได้จากการสืบค้น
5. Use case "Define Query" : จัดการการกำหนดคำสั่งการสืบค้นข้อมูล
6. Use case "Execute Query" : จัดการสั่งการให้ตัวแทนของ iBroker ทำการสืบค้นข้อมูล
7. Use case "Retrieve Data" : จัดการการรับผลการสืบค้นข้อมูล ทั้งในรูปแบบของ "การส่งผลในคราวเดียวกัน" และ "การทยอยการส่งผล"

Actor

1. Developer: ผู้พัฒนาแอพลิเคชั่นบนพื้นฐานของกริดสารสนเทศ
2. iBroker

ทั้งนี้ สามารถแสดงรายละเอียดของแต่ละ use case ได้ดังตารางที่ 3-1 ถึง 3-7

Use case name	Instantiate iBroker via Immediate Command
Participating actors	ถูกเรียกใช้งานโดยแอพลิเคชั่นที่มาติดต่อด้วย
Entry Condition	1. รับ endpoint URI สำหรับติดต่อกับ factory service ของ iBroker และรูปแบบการสืบค้นข้อมูล "การส่งผลในคราวเดียวกัน"
Flow of events	2. กำหนดค่าอ้างอิงถึง factory service ของ iBroker 3. กำหนดให้ factory service สร้าง resource 4. กำหนดค่าอ้างอิงถึงตัวแทนของ iBroker ผ่านทาง factory service และ resource 5. เตรียมการให้ตัวแทนของ iBroker รองรับการส่งผลการสืบค้นข้อมูลในรูปแบบของ "การส่งผลในคราวเดียวกัน"
Exit Condition	6. ส่งค่าอ้างอิงถึงตัวแทนของ iBroker แก่แอพลิเคชั่นที่มาติดต่อด้วย

ตารางที่ 3-1 Use case "Instantiate iBroker via Immediate Command"

Use case name	Instantiate iBroker via Continuous Command
Participating actors	ถูกเรียกใช้งานโดยแอพลิเคชั่นที่มาติดต่อด้วย
Entry Condition	1. รับ endpoint URI สำหรับติดต่อกับ factory service ของ iBroker และรูปแบบการสืบค้นข้อมูล “การทยอยการส่งผล”
Flow of events	2. กำหนดค่าอ้างอิงถึง factory service ของ iBroker 3. กำหนดให้ factory service สร้าง resource 4. กำหนดค่าอ้างอิงถึงตัวแทนของ iBroker ผ่านทาง factory service และ resource 5. เตรียมการให้ตัวแทนของ iBroker รองรับการส่งผลการสืบค้นข้อมูลในรูปแบบของ “การทยอยการส่งผล”
Exit Condition	6. ส่งค่าอ้างอิงถึงตัวแทนของ iBroker แก่แอพลิเคชั่นที่มาติดต่อด้วย

ตารางที่ 3-2 Use case “Instantiate iBroker via Continuous Command”

Use case name	Manage Resource and Status
Participating actors	ถูกเรียกใช้งานโดย use case “Instantiate iBroker”
Entry Condition	-
Flow of events	1. ตรวจสอบสถานะการทำงานของ iBroker ซึ่ง • ถ้า iBroker มีการปรับปรุง resource จะส่งผลการสืบค้นข้อมูลไปยัง use case “Manage Event” ผ่าน use case “Instantiate iBroker” • ถ้า iBroker มีสถานะเป็น “สำเร็จ” จะแจ้งให้ยังแอพลิเคชั่นที่มาติดต่อ
Exit Condition	2. แจ้งผลไปยัง use case “Manage Event” หรือ แอพลิเคชั่นที่มาติดต่อ

ตารางที่ 3-3 Use case “Manage Resource and Status”

Use case name	Manage event
Participating actors	ถูกเรียกใช้งานโดย use case “Instantiate iBroker”
Entry Condition	1. รับผลการสืบค้นจาก use case “Manage Resource and Status” ผ่านทาง use case “Instantiate iBroker”
Flow of events	2. สร้าง BrokerEvent ซึ่งบรรจุผลการสืบค้นข้อมูล
Exit Condition	3. ส่ง BrokerEvent ที่ได้สร้างขึ้นกลับไปยังโปรแกรมที่มาติดต่อด้วย

ตารางที่ 3-4 Use case “Manage Event”

Use case name	Define Query
Participating actors	ถูกเรียกใช้งานโดยแอพลิเคชั่นที่มาติดต่อด้วย
Entry Condition	-
Flow of events	1. อำนวยความสะดวกในการให้ผู้พัฒนาฯสร้างคำสั่งการสืบค้นข้อมูลด้วย code assist ที่มากับ IDE ซึ่งนำเสนอการสร้างคำสั่งในลักษณะที่คล้ายกับ SQL 2. แปลงคำสั่งการสืบค้นข้อมูลนั้นๆให้อยู่ในรูปของ InformationQuery 3. เก็บค่า InformationQuery ไว้ในตัวแทนของ iBroker
Exit Condition	-

ตารางที่ 3-5 Use case “Define Query”

Use case name	Execute Query
Participating actors	ถูกเรียกใช้งานโดยแอพลิเคชั่นที่มาติดต่อด้วย
Entry Condition	1. แอพลิเคชั่นเรียกใช้คำสั่ง executeQuery()
Flow of events	2. ตัวแทน iBroker เรียกใช้คำสั่ง getInformation() ของ iBroker พร้อมทั้งแนบค่า InformationQuery ไปยังคำสั่ง getInformation() ด้วย ทั้งนี้เพื่อให้ iBroker ดำเนินการสืบค้นข้อมูล 3. ตัวแทน iBroker จัดการตรวจสอบสถานะการทำงานของ iBroker ผ่านทาง use case “Manage resource and status”
Exit Condition	-

ตารางที่ 3-6 Use case “Execute Query”

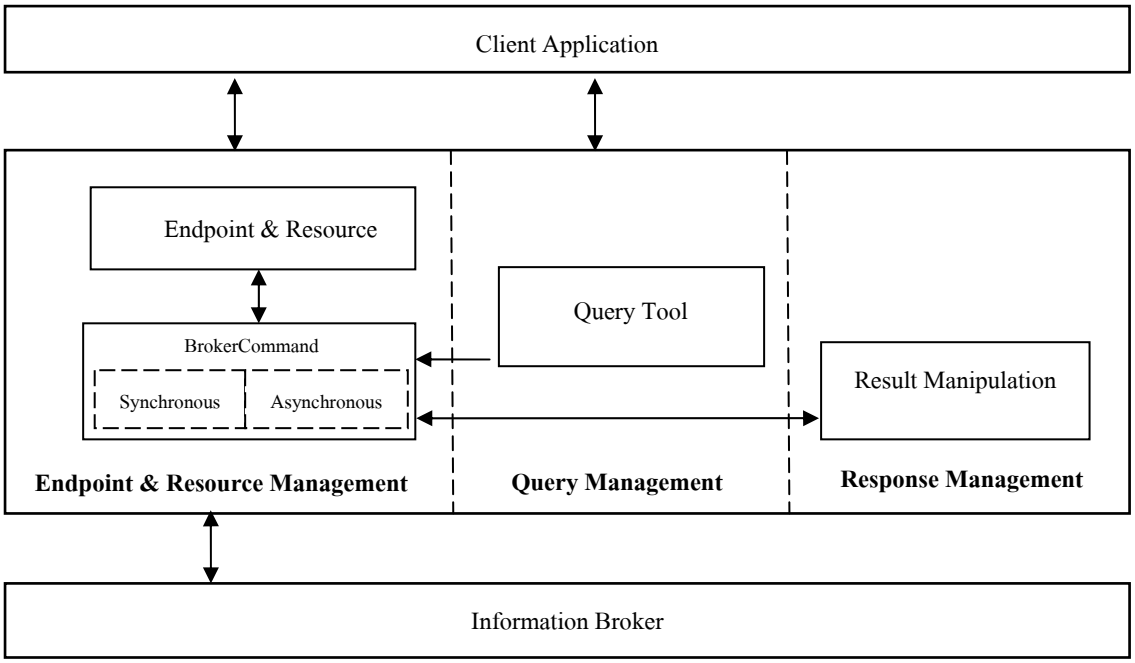
Use case name	Retrieve Query
Participating actors	ถูกเรียกใช้งานโดยแอพลิเคชั่นที่มาติดต่อด้วย
Entry Condition	1. แอพลิเคชั่นเรียกใช้คำสั่ง getDataElemenet()
Flow of events	2. แอพลิเคชั่นได้มาซึ่งข้อมูลจากการสืบค้นในรูปของเอกสาร XML
Exit Condition	-

ตารางที่ 3-7 Use case “Retreive Query”

4.สถาปัตยกรรม และโครงสร้างของ Information Grid Client API

4.1 สถาปัตยกรรมของ Information Grid Client API

รูปที่ 4-1 แสดงสถาปัตยกรรมของ Information Grid Client API ซึ่งแบ่งการทำงานออกเป็น 3 ส่วนหลักๆคือ ส่วนบริหารจัดการการติดต่อกับ iBroker (iBroker Command Management) ส่วนบริหารจัดการคำสั่งการสืบค้นข้อมูล (Query Management) และส่วนบริหารจัดการผลการสืบค้นข้อมูล (Response Management) โดยมีรายละเอียด ดังนี้



รูปที่ 4-1 สถาปัตยกรรมของ Information Grid Client API

ส่วนบริหารจัดการการติดต่อกับ iBroker (iBroker Command Management) เป็นส่วนที่ทำหน้าที่ในการกำหนดค่าอ้างอิงที่เป็นตัวแทนของ iBroker และทำการติดต่อกับ iBroker เพื่อสั่งการให้ iBroker ทำการสืบค้นข้อมูล และส่งผลการสืบค้นข้อมูลมาให้ โดยมีองค์ประกอบที่สำคัญอยู่ 2 โมดูลคือ

- Endpoint & Resource Module ทำหน้าที่ในการกำหนดค่าอ้างอิงที่เป็นตัวแทนของ iBroker ซึ่งมีกระบวนการดังนี้ (1) กำหนดค่าอ้างอิงถึง factory service ของ iBroker (2) กำหนดให้ factory service สร้าง resource ขึ้นมาต่อการเรียกใช้งาน iBroker หนึ่งๆ (3) กำหนดค่าอ้างอิงถึง iBroker ผ่านทาง factory service และ resource ที่ได้สร้างขึ้น และ (4) กำหนดค่าอ้างอิงที่เป็นตัวแทนของ iBroker ผ่านทางค่าอ้างอิงของ iBroker โดยตัวแทนของ iBroker ในที่นี้คือ BrokerCommand

- BrokerCommand Module ทำหน้าที่ในการ (1) จัดการสั่งการให้ iBroker ทำการสืบค้นข้อมูล และ (2) จัดการการส่งออกผลการสืบค้นข้อมูลในรูปแบบ "การส่งผลในคราวเดียวกัน" (synchronous response) และ "การทยอยการส่งผล" (asynchronous response) ซึ่งในการทยอยการส่งผลนั้น BrokerCommand Module จะทำการมอนิเตอร์สถานะการทำงานของ iBroker และตรวจสอบผลการสืบค้นผ่านทาง resource ที่สร้างขึ้นมาต่อการใช้งาน iBroker หนึ่งๆ โดยเมื่อ iBroker มีบันทึกผลใหม่ใน resource ส่วนการทำงานนี้จะทำการร้องขอผลการสืบค้นนี้จาก iBroker และส่งผลที่ได้กลับไปยังแอพลิเคชั่นผ่านทางกลไกของ event-driven programming

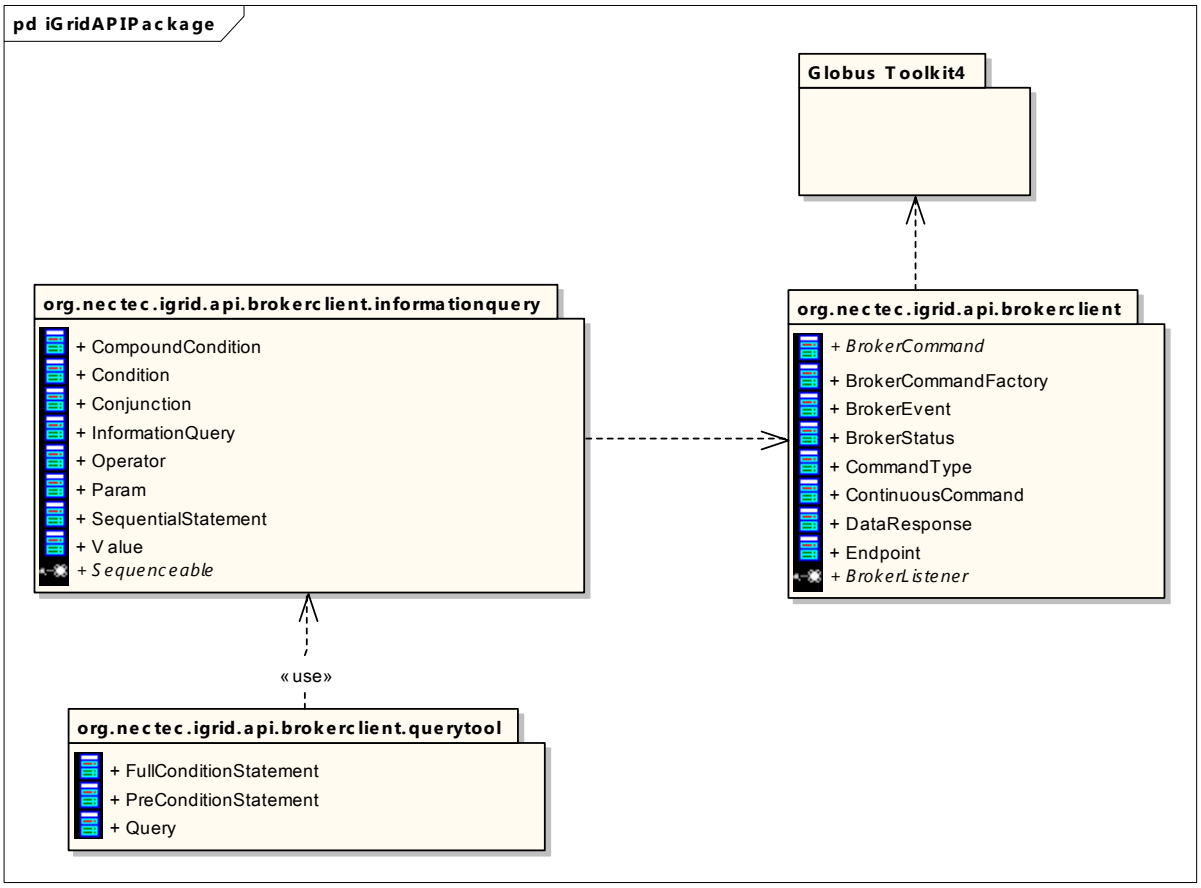
ส่วนบริหารจัดการคำสั่งการสืบค้นข้อมูล (Query Management) เป็นส่วนที่ทำหน้าที่อำนวยความสะดวกในการสร้างคำสั่งการสืบค้นข้อมูล และแปลงคำสั่งให้อยู่ในรูปแบบที่ iBroker ต้องการ โดยมีองค์ประกอบที่สำคัญอยู่ 1 โมดูลคือ

- Query Tool Module เป็น Programming Interface ที่อำนวยความสะดวกให้แก่ผู้พัฒนาฯทำการสร้างคำสั่งการสืบค้นข้อมูล ซึ่งมีรูปแบบที่สอดคล้องกับภาษา SQL และ CQL ที่ผู้พัฒนาส่วนใหญ่คุ้นเคย ทั้งนี้คำสั่ง SQL และ CQL นี้จะถูกแปลงให้อยู่ในรูปของ InformationQuery เพื่อส่งให้ iBroker ทำการสืบค้นข้อมูลต่อไป

ส่วนบริหารจัดการผลการสืบค้นข้อมูล (Response Management) เป็นส่วนที่ทำหน้าที่ในการจัดรูปแบบของผลการสืบค้นข้อมูลให้อยู่ในรูปแบบที่สะดวกต่อการนำไปใช้งาน ซึ่งจัดการโดย Result Manipulation Module

4.2 Package diagram

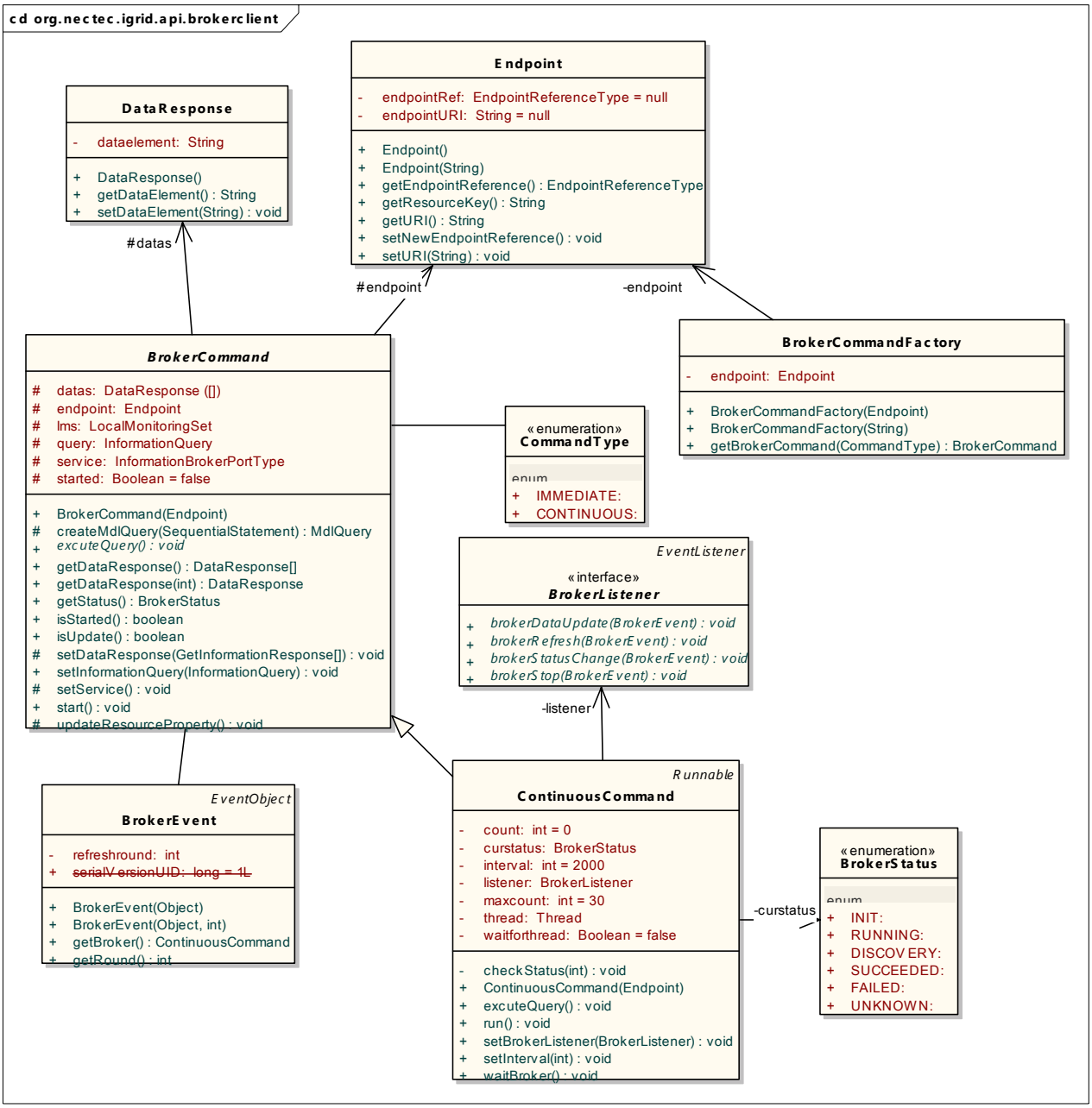
Information Grid Client API แบ่งส่วนการทำงานต่างๆออกเป็น 3 แพคเกจ ดังแสดงในรูปที่ 4-2 ซึ่งประกอบด้วย BrokerClient, InformationQuery และ QueryTool โดยแพคเกจ BrokerClient ได้ครอบคลุมการทำงานในส่วนบริหารจัดการการติดต่อกับ iBroker และส่วนบริหารจัดการผลการสืบค้นข้อมูล ในขณะที่ แพคเกจ InformationQuery และ QueryTool จะครอบคลุมการทำงานในส่วนบริหารจัดการคำสั่งการสืบค้นข้อมูล



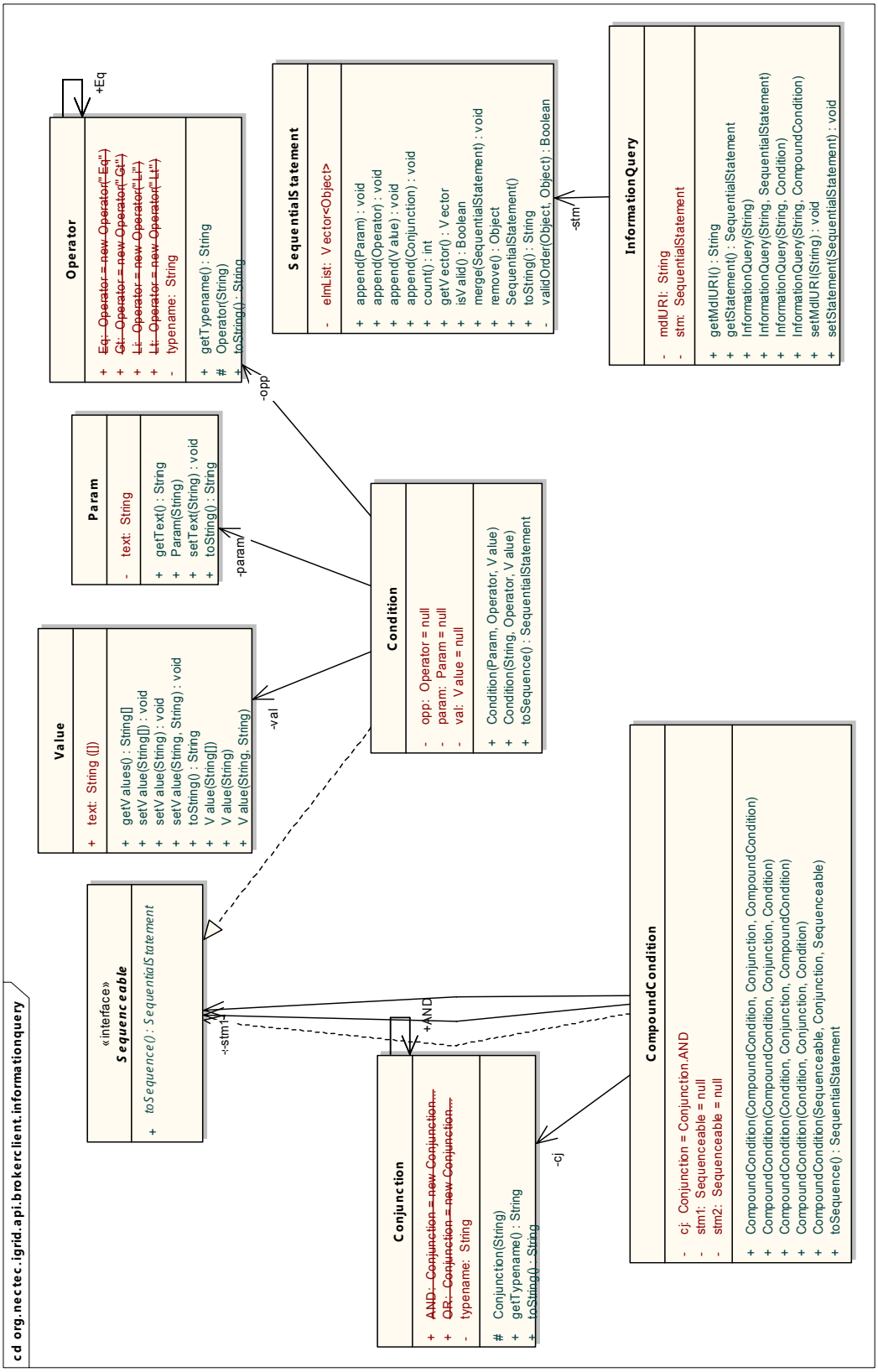
รูปที่ 4-2 Package diagram ของ Information Grid Client API

4.3 Class diagram

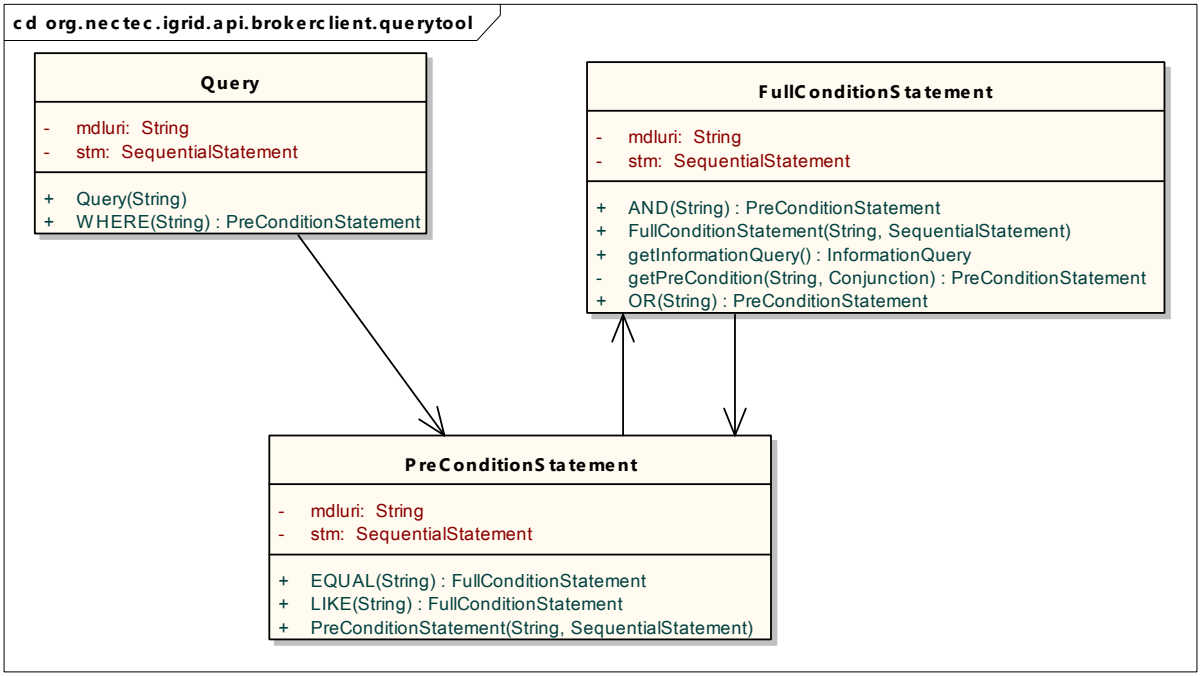
รูปที่ 4-3 ถึง 4-5 แสดงถึง class diagram สำหรับแพคเกจ BrokerClient , InformationQuery และ QueryTool ตามลำดับ



รูปที่ 4-3 Class Diagram สำหรับแพคเกจ Broker Client



รูปที่ 4-4 Class Diagram สำหรับแพคเกจ Information Query



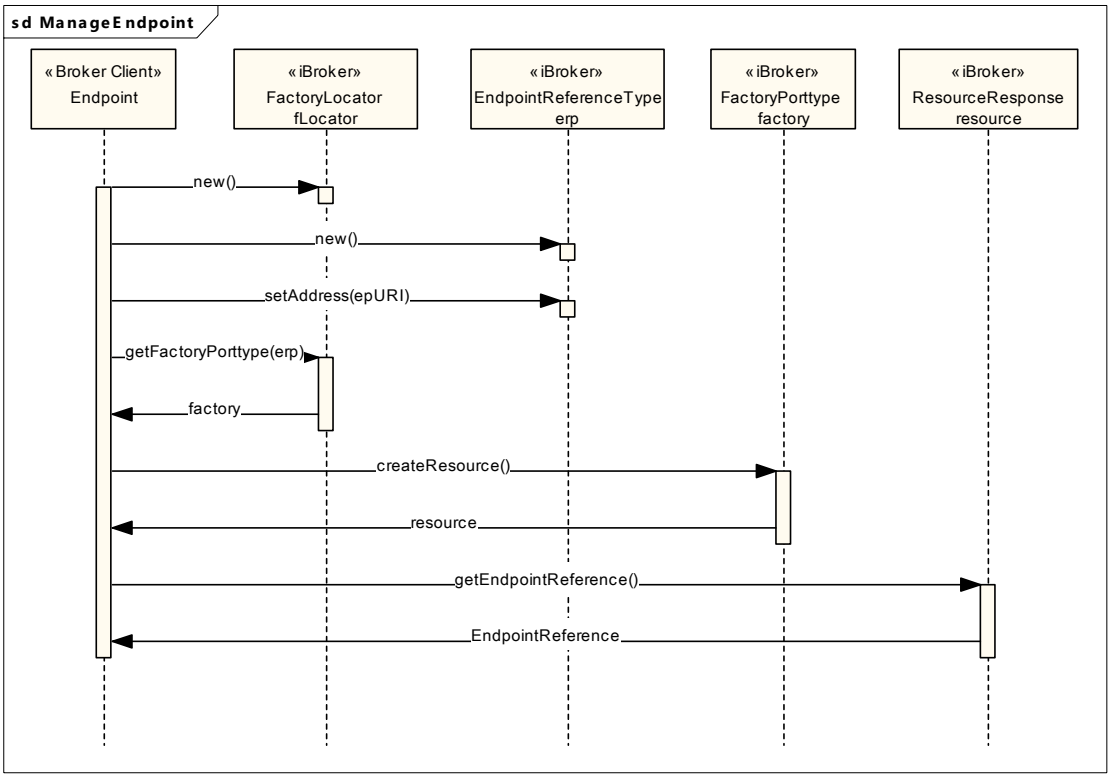
รูปที่ 4-5 Class Diagram สำหรับแพคเกจ QueryTool

4.4 Sequence diagram

4.4.1 การจัดการ Endpoint

รูปที่ 4-6 แสดงถึง sequence diagram สำหรับการได้มาซึ่งค่าอ้างอิงถึงตัวแทนของ iBroker ทั้งนี้ เนื่องจากพื้นที่ของกระดาษมีขนาดจำกัด จึงขอใช้ตั้งชื่อ class ใหม่เพื่ออ้างอิงถึง class ที่มีอยู่ในระบบ ดังนี้

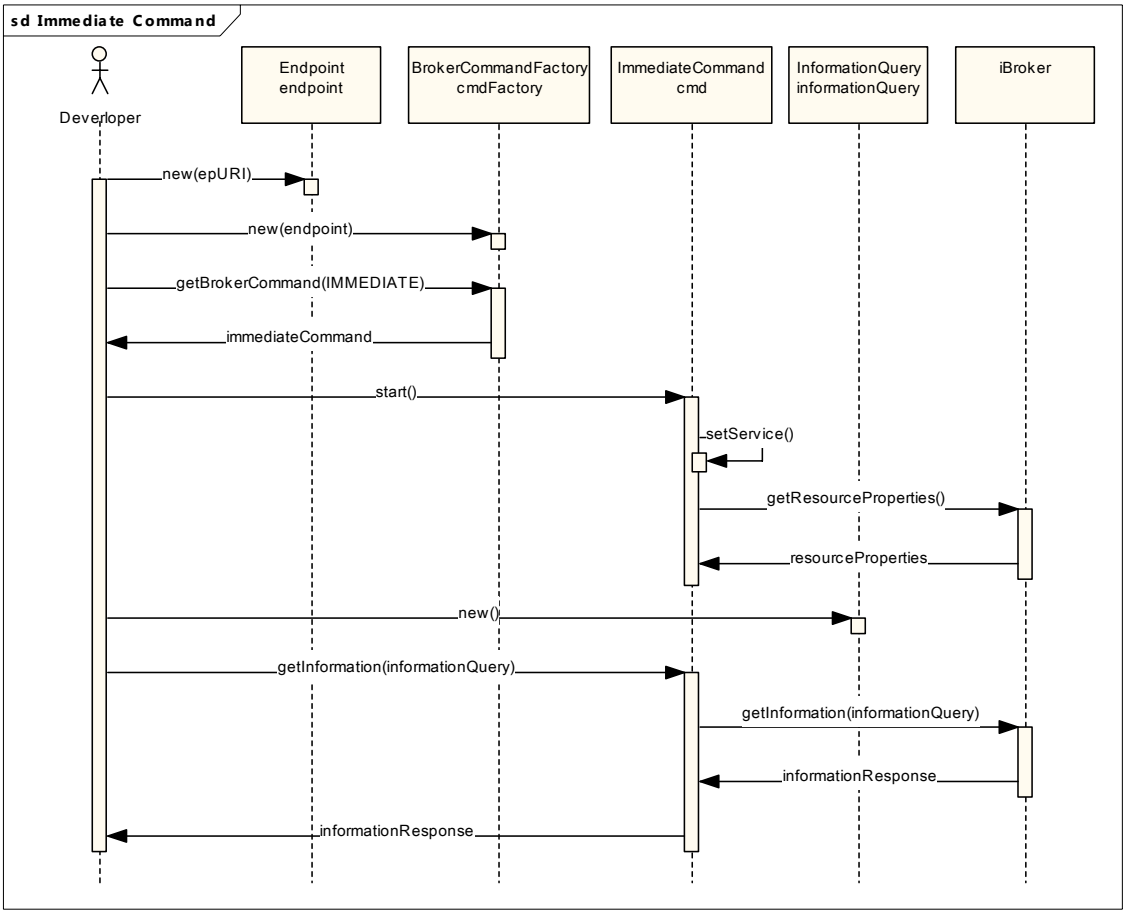
ชื่อ class เดิม	ชื่อ class ที่อยู่ในรูปที่ 4-6
InformationBrokerFactoryServiceAddressingLocator	iBFactoryLocator
EndpointReferenceType	EPR
InformationBrokerFactoryPortType	iBFactoryPortType
CreateResourceResponse	ResourceResponse
InformationBrokerPortType	iBPortType



รูปที่ 4-6 Sequence Diagram สำหรับการ Manage Endpoint

4.4.2 การเรียกคำสั่งแบบ Immediate Command

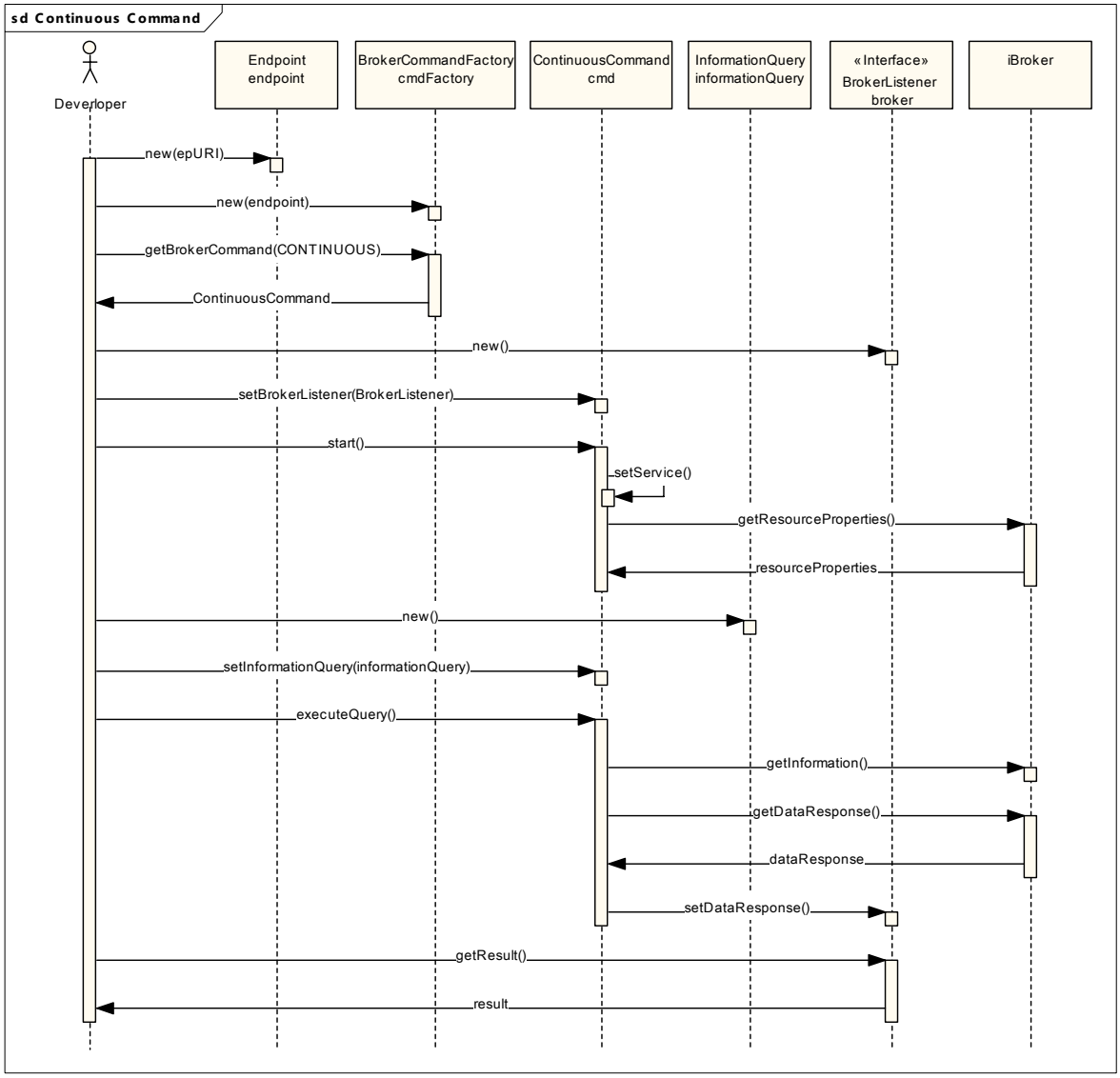
รูปที่ 4-7 แสดงถึง sequence diagram สำหรับการติดต่อการ iBroker โดยให้ iBroker ส่งผลการสืบค้นข้อมูลในรูปแบบของ “การส่งผลในคราวเดียวกัน” (Immediate Command)



รูปที่ 4-7 Sequence Diagram สำหรับการติดต่อInformation Broker แบบ Immediate Command

4.4.3 การเรียกคำสั่งแบบ Continuous Command

รูปที่ 4-7 แสดงถึง sequence diagram สำหรับการติดต่อการ iBroker โดยให้ iBroker ส่งผลการสืบค้นข้อมูลในรูปแบบของ “การทยอยการส่งผล” (Continuous Command)



รูปที่ 4-8 Sequence Diagram สำหรับการติดต่อInformation Broker แบบ Continuous Command

5.การเขียนโปรแกรมด้วย Information Grid Client API

5.1 การเขียนโปรแกรมเพื่อรับผลการสืบค้นข้อมูลในรูปแบบ “การส่งผลในคราวเดียวกัน”

รูปที่ 5-1 แสดงถึงตัวอย่างการเขียนโปรแกรมด้วย iGrid Client API เพื่อให้ส่งผลการสืบค้นข้อมูลในรูปแบบ “การส่งผลในคราวเดียวกัน” โดยมีขั้นตอน 1 ขั้นตอนดังนี้

1. สร้างโปรแกรม TestImmediateCommand.java ซึ่งเป็นโปรแกรมหลักที่มีการตั้งค่าอ้างอิงถึงตัวแทน iBroker, กำหนดคำสั่งการสืบค้นข้อมูล และสั่งการให้ทำการสืบค้นข้อมูล ทั้งนี้โปรแกรมนี้จะทำการติดต่อไปยัง iBroker ที่มี endpoint “http://127.0.0.1:8080/wsrf/services/igrid/InformationBrokerFactoryService” เพื่อให้ทำการสืบค้นข้อมูลบนพื้นฐานมาตรฐานโครงสร้างข้อมูลที่กำหนดไว้ใน http://www.thai-research.net/mdl/complex-type/researchproject โดยมี “project/name like ‘ข้าว’ and project/name like ‘วิจัย’” เป็นคำสั่งในการสืบค้นข้อมูล

```
import org.nectec.igrid.api.brokerclient.informationquery.*;
import org.nectec.igrid.api.brokerclient.*;
import org.nectec.igrid.api.brokerclient.querytool.*;
public class TestCommand {
  public static void main(String args[]) throws Exception
  {
        String mdlURI = "http://www.thai-research.net/mdl/complextype/researchproject";
        Endpoint ep = new Endpoint("http://127.0.0.1:8080/wsrf/services/igrid/InformationBrokerFactoryService");
        BrokerCommandFactory factory = new BrokerCommandFactory(ep);
        ImmediateCommand cmd =
                (ImmediateCommand)factory.getBrokerCommand(CommandType.IMMEDIATE);
        cmd.start();
        cmd.setInformationQuery(new Query(mdlURI).WHERE("Project/name").LIKE("ข้าว").
                AND("Project/name").LIKE("วิจัย").getInformationQuery());
        cmd.excuteQuery();
        String result = cmd.getDataResponse().getDataElement();
  }
```

import class ที่เกี่ยวข้อง

กำหนดค่าอ้างอิงถึงตัวแทน iBroker

กำหนดคำสั่งการสืบค้นข้อมูล

เริ่มต้นการสืบค้นข้อมูล

รับผลการสืบค้นข้อมูล

รูปที่ 5-1 ตัวอย่างโปรแกรม TestImmediateCommand.java

5.2 การเขียนโปรแกรมเพื่อรับผลการสืบค้นข้อมูลในรูปแบบ “การทยอยการส่งผล”

รูปที่ 5-2 และ 5-3 แสดงถึงตัวอย่างการเขียนโปรแกรมด้วย iGrid Client API เพื่อให้ส่งผลการสืบค้นข้อมูลในรูปแบบ “การทยอยการส่งผล” โดยการเขียนโปรแกรมจะอยู่ในลักษณะของ Event-Driven Programming โดยมีขั้นตอนดังนี้

1. สร้างโปรแกรม TestContinuousCommand.java ซึ่งเป็นโปรแกรมหลักที่มีการตั้งค่าอ้างอิงถึงตัวแทน iBroker, กำหนดคำสั่งการสืบค้นข้อมูล และสั่งการให้ทำการสืบค้นข้อมูล ตลอดจนการตั้งค่าโปรแกรมเพื่อใช้ในการดักฟัง Event ทั้งนี้โปรแกรมนี้จะทำการติดต่อไปยัง iBroker ที่มี endpoint “http://127.0.0.1:8080/wsrf/services /igrid/Information BrokerFactoryService” เพื่อให้ทำการสืบค้นข้อมูลบนพื้นฐานมาตรฐานโครงสร้างข้อมูลที่กำหนดไว้ใน http://www.thai-research.net/mdl/complex-type/researchproject โดยมี “project/name like ‘ข้าว’ and project/name like ‘วิจัย’” เป็นคำสั่งในการสืบค้นข้อมูล

```
import org.nectec.igrid.api.brokerclient.informationquery.*;
import org.nectec.igrid.api.brokerclient.*;
import org.nectec.igrid.api.brokerclient.querytool.*;
public class TestCommand {
  public static void main(String args[]) throws Exception
  {
        String mdlURI = "http://www.thai-research.net/mdl/complextype/researchproject";
        Endpoint ep = new Endpoint("http://127.0.0.1:8080/wsrf/services/igrid/InformationBrokerFactoryService");
        BrokerCommandFactory factory = new BrokerCommandFactory(ep);
        ContinuousCommand cmd =
                (ContinuousCommand)factory.getBrokerCommand(CommandType.CONTINUOUS);
        cmd.setBrokerListener(new MyBroker());
        cmd.start();
        cmd.setInformationQuery(new Query(mdlURI).WHERE("Project/name").LIKE("ข้าว").
                AND("Project/name").LIKE("วิจัย").getInformationQuery());
        cmd.excuteQuery();
  }
```

import class ที่เกี่ยวข้อง

กำหนดค่าอ้างอิงถึงตัวแทน iBroker

กำหนด BrokerListener เพื่อดักฟังเหตุการณ์ (event)

กำหนดคำสั่งการสืบค้นข้อมูล

เริ่มต้นการสืบค้นข้อมูล

รูปที่ 5-2 ตัวอย่างโปรแกรม TestContinuousCommand.java

2. สร้างโปรแกรม MyBroker.java ซึ่งเป็นโปรแกรมสำหรับติดตามสถานะการทำงานของ iBroker ตลอดจนการรับผลการสืบค้นข้อมูลในรูปแบบของ “การทยอยการส่ง” โดยโปรแกรมนี้จะต้องเขียนให้สอดคล้องกับ interface BrokerListener (สามารถศึกษารายละเอียดของการดักจับเหตุการณ์ได้ในหัวข้อถัดไป)

```
import org.nectec.igrid.api.brokerclient.*;
import java.io.*;
public class MyBroker implements BrokerListener{
  public MyBroker(){ }
  public void brokerDataUpdate(BrokerEvent e) {
     PrintStream out = new PrintStream(new FileOutputStream("D:\\OutFile.xml"));
     out.println(e.getBroker().getDataResponse(0).getDataElement());
  }
  public void brokerStatusChange(BrokerEvent e) {
     System.out.println(String.valueOf(e.getRound()) + " " +
              ((ContinuousCommand)e.getSource()).getStatus().toString());
  }
  public void brokerRefresh(BrokerEvent e){}
  public void brokerStop(BrokerEvent e){}
}
```

Implement BrokerListener interface

รับผลการสืบค้นข้อมูล

รูปที่ 5-3 ตัวอย่างโปรแกรม MyBroker.java

5.3 การดักจับเหตุการณ์ (Events) ด้วย BrokerListener

```
public class MyBroker implements BrokerListener{

        public MyBroker(){ }

        public void brokerDataUpdate(BrokerEvent e){}

        public void brokerStatusChange(BrokerEvent e){}

        public void brokerRefresh(BrokerEvent e){}

        public void brokerStop(BrokerEvent e){}

}
```

รูปที่ 5-3 BrokerListener interface

รูปที่ 5-3 แสดง BrokerListener interface ซึ่งกำหนดเหตุการณ์เพื่อรองรับการทำงานของ iGrid client API ไว้อยู่ 4 เหตุการณ์ด้วยกัน โดยมีรายละเอียดดังนี้

- เหตุการณ์ brokerDataUpdate
  เป็นเหตุการณ์ที่เกิดขึ้นเมื่อ iBroker ได้รับข้อมูลจาก iService และข้อมูลนั้นอยู่ในสภาพพร้อมที่จะส่งให้โปรแกรมที่ติดต่อ ด้วยการเรียกใช้คำสั่ง

  <BrokerEvent object>.getBroker()getDataResponse().getDataElement()

- เหตุการณ์ brokerStatusChange
  เป็นเหตุการณ์ที่เกิดขึ้นเมื่อสถานะการทำงานของ InformationBroker มีการเปลี่ยนแปลง โดยสามารถรับค่าสถานะ ด้วยการเรียกใช้คำสั่ง

  <BrokerEvent.object>.getSource()).getStatus()

- เหตุการณ์ brokerRefresh
  เป็นเหตุการณ์ที่อำนวยความสะดวกให้แก้ผู้พัฒนาฯสามารถเขียนโปรแกรมเพื่อมอนิเตอร์การทำงานของ iBroker ซึ่งสามารถกระทำได้ด้วยคำสั่ง

  <BrokerEvent.object>.getRound()

- เหตุการณ์ brokerStop
  เป็นเหตุการณ์ที่เกิดขึ้นเมื่อ iBroker สิ้นสุดการทำงานตามคำสั่งใดๆ ซึ่งอาจจะหยุดการทำงานแบบปกติหรือหยุดการทำงานเพราะเกิด Error ก็ได้

6.บทสรุป

บันทึกวิจัยฉบับนี้ได้นำเสนอถึงการออกแบบ และพัฒนา Information Grid API ซึ่งได้รับการพัฒนาขึ้นมาโดยมีวัตถุประสงค์หลักเพื่อซ่อนความซับซ้อนในการเขียนโปรแกรม และ การติดตั้ง GT4 ตลอดจนเพื่อสนับสนุนและส่งเสริมให้มีการพัฒนาแอพลิเคชั่นบนพื้นฐานของกริดสารสนเทศ ทั้งนี้ iGrid Client API ที่ได้พัฒนาขึ้นนั้นได้มุ่งเน้นไปยังการทำงานใน 3 ด้านหลัก คือ การกำหนดค่าอ้างอิงของ iBroker, การสร้างคำสั่งการสืบค้นข้อมูล และ การรับผลการสืบค้นข้อมูลจาก iBroker อย่างไรก็ดี ผลการสืบค้นข้อมูลที่ได้จาก iGrid Client API นี้ยังคงอยู่ในรูปของเอกสาร XML ซึ่งผู้พัฒนาฯจะต้องทำการสกัดเพื่อให้ได้มาซึ่งข้อมูลตามเอลิเมนต์ที่ต้องการ

ดังนั้น เพื่อสนับสนุนให้ผู้พัฒนาฯสามารถได้มาซึ่งข้อมูลตามเอลิเมนต์ที่ต้องการผ่านทางการเขียนโปรแกรมในลักษณะของ object-oriented programming (ไม่ต้องทำการสกัด) ได้นั้น หน่วยปฏิบัติการฯจึงมีแผนที่จะทำการพัฒนา plug-in เพื่อแปลงมาตรฐานโครงสร้างข้อมูลใดๆ ให้อยู่ในรูป Object model ที่สอดคล้องกับมาตรฐานหนึ่งๆ โดย plug-in นี้จะสามารถทำงานร่วมกับ Integrated Development Environment (IDE) ต่างๆได้