บทที่ 2

การโปรแกรมเชิงมหภาค

2.1 รูปแบบการเขียนโปรแกรมที่เหมาะสม

โดยทั่วไปแล้วมีรูปแบบในการเขียนโปรแกรมหลักอยู่สองรูปแบบในวงการคอมพิวเตอร์ นั่นคือ การโปรแกรมเชิงประกาศ (Declarative Programming) และการโปรแกรมเชิงบังคับ (Imperative Programming) สำหรับการโปรแกรมเชิงประกาศนั้นจะ Abstract รายละเอียดของอัลกอริธึ่มได้ทั้งหมด โปรแกรมเมอร์เพียงแค่ระบุสิ่งที่ตนต้องการเท่านั้นโดยไม่ต้องระบุว่าจะทำอย่างไรจึงจะได้ผลลัพธ์มา ตัวแปลภาษาจะทำการเติมอัลกอริธึ่มที่จำเป็นต้องใช้ให้เอง การสร้างรายละเอียดในเชิงอัลกอริธึ่มให้อย่างอัตโนมัติเช่นนี้อาจมีประสิทธิภาพได้สำหรับงานง่ายๆเฉพาะอย่างเท่านั้น เช่น ฐานข้อมูล แต่อาจทำได้ไม่ดีนักในงานประเภทอื่น ตัวอย่างของแนวทางการเขียนโปรแกรมเชิงประกาศมีอยู่บ้างพอสมควร เช่น งานที่มีพื้นฐานมาจากภาษา SQL ได้แก่ Cougar [1] และ TAG [19] ถึงแม้ว่าการโปรแกรมเชิงประกาศจะมีข้อดีในแง่ของความง่ายในการเขียนโปรแกรม แต่ก็ไม่ได้หมายความว่าการโปรแกรมเชิงประกาศจะเหมาะสมสำหรับทุกโปรแกรมประยุกต์ในเครือข่ายไร้สายของระบบฝังตัวแต่อย่างใด

การโปรแกรมเชิงบังคับอาจจะมีความเหมาะสมมากกว่าโดยเฉพาะอย่างยิ่งในงานที่มีความซับซ้อน ซึ่งไม่อาจสร้าง Code ที่มีประสิทธิภาพให้ได้อย่างอัตโนมัติ หรือไม่อาจสร้างให้ได้โดยง่าย ยกตัวอย่างเช่น คงเป็นไปไม่ได้หรือยากมากที่จะใช้ภาษา SQL ในการเขียนหรือสร้าง Kalman Filter หรืออัลกอริธึ่ม Maximum Likelihood สำหรับการประมาณตำแหน่งวัตถุ เป็นต้น เพราะว่าภาษา SQL ไม่ได้ถูกออกแบบมาเพื่อบรรยายรายละเอียดของอัลกอริธึ่ม

การโปรแกรมเชิงประกาศและการโปรแกรมเชิงบังคับต่างก็มีข้อดีคนละด้าน หากใช้ให้เหมาะสม นอกจากนี้ การโปรแกรมทั้งสองแบบสามารถส่งเสริมกันได้ เพราะต่างทำงานได้ดีในด้านที่เป็นข้อด้อยของอีกฝ่ายหนึ่ง หากใช้ร่วมกันน่าจะสามารถกำจัดข้อด้อยของทั้งสองแบบลงได้ การบูรณาการการโปรแกรมเชิงประกาศและเชิงบังคับเข้าด้วยกันน่าที่จะก่อให้เกิดแนวทางการเขียนโปรแกรมที่ทรงพลังเหมาะสมในการ

โปรแกรมหลากหลายด้าน ในโครงการนี้ เราเสนอว่าการบูรณาการดังกล่าวนั้นทำได้ถ้าการโปรแกรมเชิงประกาศนั้นถูกใช้เฉพาะในบางส่วนของโปรแกรมเท่านั้น ไม่ใช่ทั้งหมด

โดยทั่วไปแล้ว ส่วนของโปรแกรมที่ควร Abstract คือ

1. ส่วนที่ไม่เกี่ยวข้องกับโปรแกรมหลัก
2. ส่วนที่ต้องมีในโปรแกรมประยุกต์ทั่วไป
3. ส่วนที่ซ้ำซากน่าเบื่อหน่ายสำหรับโปรแกรมเมอร์

การจะระบุว่าส่วนไหนของโปรแกรมน่าที่จะต้องทำการ Abstract ได้ จำเป็นอย่างยิ่งที่จะต้องทำความเข้าใจโปรแกรมประยุกต์ของเครือข่ายไร้สายของระบบฝังตัวให้ดีเสียก่อน โดยปกติแล้ว โปรแกรมเป็นชุดของการกระทำการบนตัวแปรและทรัพยากร เนื่องจากตัวแปรมักจะถูกเข้าถึงบ่อยครั้งกว่า ภาษาสำหรับการเขียนโปรแกรมจึงมักมีกระบวนการในการเข้าถึงตัวแปรที่ง่ายกว่าการเข้าถึงทรัพยากร

จึงไม่น่าแปลกใจนักที่ การเข้าถึงทรัพยากรโดยปกติจะยุ่งยากกว่า โดยเฉพาะอย่างยิ่งในระบบเครือข่ายที่แยกความแตกต่างชัดเจนระหว่างทรัพยากรของโหนดเอง และทรัพยากรของโหนดอื่นที่อยู่ไกลออกไป ตามธรรมดาแล้วทรัพยากรมักจะถูกผูกเข้ากับโหนดไว้ตั้งแต่ต้น ดังนั้นในการระบุทรัพยากรของโหนดอื่นจำเป็นต้องใช้หมายเลขเครื่องของโหนดนั้นร่วมด้วย ถ้าเราไม่รู้หมายเลขเครื่อง เรามักจำเป็นต้องทำกระบวนการค้นหาทรัพยากร ด้วยเหตุนี้ โปรแกรมเมอร์จึงต้องจัดการในรายละเอียดโปรแกรมจำนวนมาก ไม่ว่าจะเป็นการเชื่อมต่อ การค้นหาทรัพยากร และการเข้าถึงทรัพยากร เป็นต้น

หากเป็นเครือข่ายไร้สายของระบบฝังตัว ความยุ่งยากนี้ยิ่งมีมากขึ้นเพราะว่าทรัพยากรที่สนใจจะถูกระบุโดยคุณสมบัติในขณะทำงานแทนที่จะเป็นหมายเลขเครื่อง ยกตัวอย่างเช่น เราอาจต้องการเข้าถึงเซ็นเซอร์ที่วัดอุณหภูมิได้เกิน 30 องศาเซลเซียสเท่านั้น ในกรณีนี้ การค้นหาทรัพยากรในเครือข่ายไร้สายของระบบฝังตัวเป็นเรื่องจำเป็น ปกติสามัญ แทนที่จะเป็นเรื่องที่เลือกที่จะทำหรือไม่ก็ได้ คุณสมบัติของทรัพยากรนั้นค่อนข้างมีพลวัตรสูงเพราะสภาวะแวดล้อมอาจจะมีอุณหภูมิสูงขึ้นหรือลดลงเมื่อใดก็ได้ อาจมีความแปรปรวนสูง รวมทั้งเป็นสภาวะแวดล้อมที่ไม่เป็นมิตร การผูกหรือแมป ทรัพยากรอาจถูกต้องในขณะหนึ่งเมื่อเวลาผ่านไปอาจไม่ถูกต้องแล้ว เพราะว่าทรัพยากรที่ตรงตามคุณสมบัติในขณะหนึ่ง อาจไม่ตรงตามคุณสมบัติอีกต่อไป

แต่ว่าต่อให้คุณสมบัติของทรัพยากรจะไม่เปลี่ยนแปลงก็ตาม ทรัพยากรที่ผูกไว้ก็อาจจะไม่สามารถเข้าถึงได้เพราะว่าพลวัตรของเครือข่าย เช่น การเคลื่อนที่ของโหนด เป็นต้น โปรแกรมสำหรับเครือข่ายไร้สายของระบบฝังตัวจะต้องสามารถจัดการกับการเปลี่ยนแปลง การผูกที่ไม่ถูกต้องอีกต่อไป การค้นหาทรัพยากรอื่นที่เท่าเทียมกันมาผูกแทน การผูกทรัพยากรใหม่ที่เพิ่งค้นพบและตรงตามคุณสมบัติที่ต้องการ เรื่องจุกจิกที่เกิดขึ้นได้บ่อยเหล่านี้ได้ เรื่องเหล่านี้ไม่ว่าจะเป็นการค้นหา การเข้าถึง การผูกใหม่ และการเชื่อมต่อ ล้วนแล้วแต่ซ้ำซากน่าเบื่อหน่ายสำหรับโปรแกรมเมอร์ ดังนั้น ส่วนของโปรแกรมที่เกี่ยวข้องกับทรัพยากรจึงเป็นตัวเลือกที่สมเหตุสมผลสำหรับการ Abstraction แบบเชิงประกาศของเรา

2.2 การเรียกชื่อทรัพยากรเชิงประกาศ

เพื่อทำให้การเขียนโปรแกรมสำหรับเครือข่ายไร้สายของระบบฝังตัวทำได้ง่ายขึ้น เราขอเสนอวิธีที่โปรแกรมเครือข่ายเป็นเสมือนหน่วยเดียวกัน โดยในที่นี้เราจะมองเสมือนว่าเครือข่ายทั้งเครือข่ายเป็นเพียง Abstract Machine เครื่องหนึ่ง ถึงแม้ในทางกายภาพแล้ว ทรัพยากรจะอยู่กันอย่างกระจัดกระจาย ทรัพยากรเหล่านั้นทั้งหมดล้วนแล้วแต่อยู่บนเครื่องเดียวกันในโมเดลของเรานี้ ดังนั้นในโมเดลของเรา จึงไม่มีเครือข่าย ไม่มีการเชื่อมต่อ ไม่มีคำว่าใกล้ ไม่มีคำว่าไกล

2.3 ตัวแปรทรัพยากร

การโปรแกรมเครือข่ายไร้สายของระบบฝังตัวสามารถถูกทำให้ง่ายลงได้โดยการทำให้การเข้าถึงทรัพยากรง่ายเหมือนกับการเข้าถึงตัวแปร ดังนั้น เราจึงขอเสนอ ตัวแปรพิเศษชนิดใหม่ชื่อ ตัวแปรทรัพยากร (ตัวแปรที่ถูกแมปและอ้างอิงถึงโหนดจริง) ยกตัวอย่างเช่น เราสามารถเขียนโปรแกรมเพื่ออ่านค่าเซนเซอร์แสงและควบคุมกล้องได้ดังนี้

```
Resource R, X;
printf("light intensity=%f", R->light);
X->camera=off;
```

ในตัวอย่างข้างต้น เราสมมุติให้ตัวแปรทรัพยากร R แมปไปยังโหนดที่มีเซนเซอร์แสง และตัวแปรทรัพยากร X แมปไปยังโหนดที่มีกล้อง การอ่านค่าความเข้มแสงสามารถทำได้อย่างง่ายดายโดยเพียงแค่การอ้างอิงถึงเซนเซอร์แสงของ R คือ R->light ในทางที่คล้ายๆกัน เราสามารถปิดกล้องได้โดยแค่ใส่ค่า Off ให้กับกล้องของ X คือ X->camera ไม่จำเป็นต้องมีรายละเอียดอัลกอริธึ่มสำหรับการควบคุมและการกระทำการกับทรัพยากร ตัวอย่างนี้แสดงให้เห็นว่าแนวทางของเราไม่เพียงแต่สามารถใช้ในการดึงหรือส่งข้อมูลไปยังโหนดที่ต้องการได้เท่านั้น มันยังสามารถใช้ในการควบคุมโหนดเหล่านั้นได้อีกด้วย

2.4 การจำกัดเชิงประกาศ

คงเป็นที่เข้าใจได้ว่า บางท่านอาจสงสัยว่าโหนดทางกายภาพหรือทรัพยากรไหนจะถูกแมปหรือผูกเข้ากับตัวแปรทรัพยากรกันแน่ และโปรแกรมเมอร์จะรู้เกี่ยวกับชนิดของเซนเซอร์แต่ละตัวได้อย่างไร ในที่นี้แทนที่เราจะระบุหมายเลขโหนดเหมือนอย่างในอินเทอร์เนต เราจะทำในแนวทางอื่นที่เหมาะสมกับเครือข่ายไร้สายของระบบฝังตัวมากกว่า นั้นคือการระบุคุณสมบัติที่ต้องการของโหนดเป้าหมายแทนในเชิงประกาศด้วยเงื่อนไขในเชิงตรรก ยกตัวอย่างเช่น เราสามารถระบุได้ว่า R จะต้องถูกผูกไปยังโหนดที่มีเซนเซอร์แสงที่อยู่ในป่าซึ่งมีอุณหภูมิมากกว่า 30 องศาเซลเซียส

```
Resource R = <within(location,forest)&&
                    temperature > 30 &&exist(light)>
Resource X = <a(b,c)!=0 && exist(camera)>
```

อย่างไรก็ตาม มีความเป็นไปได้อย่างสูงที่จะมีโหนดที่สอดคล้องตามคุณสมบัติข้างต้นอยู่เป็นจำนวนมาก ดังนั้นตัวแปรทรัพยากรที่แท้จริงจะถูกผูกกับเซตของโหนดที่สอดคล้องตามคุณสมบัติ มิได้ผูกอยู่กับโหนดใดโหนดหนึ่งแต่เพียงโหนดเดียว ตำแหน่งและอุณหภูมิเป็นคุณสมบัติท้องถิ่นของโหนดที่ถูกใช้ในการตัดสินว่าโหนดนั้นๆจะเป็นสมาชิกในเซตที่ผูกกับตัวแปร R หรือไม่ นอกจากนี้ เราสามารถใช้ฟังก์ชันตรรกะใดก็ได้ในเงื่อนไขเชิงประกาศ โดยผู้ใช้อาจสร้างฟังก์ชันตรรกะขึ้นมาเองก็ได้ เช่น ฟังก์ชัน a() ในตัวอย่างข้างต้น ข้อยืดหยุ่นดังกล่าวโดยทั่วไปแล้วทรงพลังมาก ควรที่จะเพียงพอต่อการกำหนดเงื่อนไขเชิงประกาศที่ซับซ้อนหลากหลายกว่าในตัวอย่างได้อีกมาก

2.5 การเข้าถึงทรัพยากร

ในที่นี้เราจะอธิบายถึงความจำเป็นที่ระบบของเราต้องสนับสนุนการเข้าถึงข้อมูลได้หลายแบบ สำหรับใช้ในสถานะการณ์ที่แตกต่างกัน โดยจะมีการวิเคราะห์ข้อดีและข้อเสียของแต่ละแบบไว้ให้ด้วยเพื่อเป็นแนวทางในการนำไปเลือกใช้ให้เหมาะสมกับงาน ในโครงการนี้ เราขอเสนอวิธี 2 วิธีในการเข้าถึงโหนดทั้งหลายที่สอดคล้องตามเงื่อนไขเชิงประกาศ คือแบบอนุกรมและแบบขนาน

- การเข้าถึงแบบอนุกรม ในแบบนี้โหนดแต่ละโหนดในเซตสามารถถูกอ้างอิงถึงได้โดยการใช้ Iterator (คล้ายกับ Iterator ใน C++ Standard Template Library) วิธีนี้ทำให้เราสามารถเข้าถึงโหนดที่สอดคล้องตามเงื่อนไขได้ทีละตัว ยกตัวอย่างเช่น เราสามารถอ่านค่าความเข้มแสงจากเซนเซอร์แสงของโหนดในเซต R ทีละตัวได้ตามลำดับดังนี้

```
Resource R;
Iterator i;
foreach i in R {
     printf("light intensity = %f\n", i->light);
}
```

อย่างไรก็ตาม การอ่านตามลำดับไม่สามารถให้ค่าที่เวลาเดียวกันของโหนดแต่ละตัวในเซตได้ เพราะความหน่วงเวลาในการเข้าถึงโหนดทุกตัวในเซตอาจมีนัยสำคัญสูง ทำให้การอ่านค่าของโหนดแรกเกิดขึ้นที่เวลาห่างจากเวลาขณะอ่านค่าของโหนดสุดท้ายมากได้ อย่างไรก็ตามการเข้าถึงแบบนี้ยังมีประโยชน์อยู่มาก เช่น การเข้าถึงโหนดแค่บางโหนดในเซต เป็นต้น

- การเข้าถึงแบบขนาน ในแบบนี้ โหนดทุกตัวในเซตจะถูกเข้าถึงพร้อมกัน การเข้าถึงแบบขนานทำได้โดยการอ้างอิงตัวแปรทรัพยากรโดยตรงดังนี้

```
Resource R;
printf("light intensity=%f", R->light);
```

ในตัวอย่างข้างต้น โปรแกรมจะแสดงค่าความเข้มของแสงของทุกโหนดในเซต R นี้ออกมาเป็นหน้าจอ ค่าความหน่วงเวลารวมทั้งหมดเมื่อใช้การเข้าถึงแบบขนานนี้จะลดลงเหลือเพียงแค่ค่าหน่วง

เวลาของการเข้าถึงโหนดตัวที่ใช้เวลามากที่สุด แนวทางนี้ไม่เพียงลดเวลาการเข้าถึงรวมเท่านั้น มันยังสามารถให้ค่าที่เวลาเดียวกันของโหนดแต่ละตัวในเซตได้ นอกจากนี้ แนวทางแบบขนานนี้ยังแตกต่างจากแนวทางแบบอนุกรมตรงที่แนวทางแบบขนานเปิดโอกาสให้มีการประมวลผลภายในเครือข่ายได้ เช่น การรวมข้อมูลในระหว่างการเดินทางของข้อมูลในเครือข่าย ซึ่งการประมวลผลในเครือข่ายนี้สามารถลดการใช้พลังงานของระบบลงได้อย่างมีนัยสำคัญ [10, 11, 13, 17, 14, 19] ตัวอย่างเช่น การใช้ฟังชัน max(A) เพื่อการหาค่ามากที่สุดในเซต A สามารถทำได้ดังนี้

```
Resource R;
printf("max light intensity = %f", max(R->light));
```

ในทางอุดมคติ ระบบควรจะส่งเฉพาะค่าที่มากที่สุดเท่านั้นเพื่อประหยัดพลังงาน โดยไม่เสียพลังงานไปในการส่งค่าๆอื่นเลย แต่ในทางปฏิบัติแล้ว การจะทำเช่นนั้นได้โหนดทุกตัวจะต้องรู้ว่าค่าที่อ่านได้ของตนเมื่อเทียบกับโหนดอื่นแล้วมีค่ามากที่สุดหรือไม่ หากมากสุดจึงส่งออกมา ซึ่งในความเป็นจริงแล้วถ้าโหนดไม่แลกเปลี่ยนค่ากันก่อน โหนดแต่ละตัวจะไม่สามารถล่วงรู้ค่าของโหนดอื่นได้ ในที่นี้เราสามารถทำให้การใช้พลังงานใกล้เคียงกับอุดมคติได้โดยการระงับการส่งหรือส่งต่อค่าที่น้อยกว่าค่าที่เคยพบมาก่อนหน้าของการเข้าถึงเดียวกัน ระหว่างการส่งต่อค่าภายในเครือข่าย การระงับการส่งต่อค่านี้จะไม่สัมฤทธิ์ผลหรืออาจเป็นไปไม่ได้ ถ้าเราทำการเข้าถึงข้อมูลแบบอนุกรมแทนที่จะเป็นแบบขนาน

2.6 การผูกทรัพยากร

โมเดลของเราสนับสนุนการผูกทรัพยากรสองประเภทคือ แบบพลวัตและแบบสถิต

- การผูกแบบพลวัต ในวิธีของเรานั้น โปรแกรมเมอร์ไม่มีความจำเป็นที่จะต้องเขียนโปรแกรมเพื่อบำรุงรักษาให้การผูกค่าระหว่างทรัพยากรทางกายภาพและตัวแปรทรัพยากรถูกต้องอยู่เสมอด้วยตนเอง เนื่องจากของคุณสมบัติของทรัพยากรเปลี่ยนแปลงค่อนข้างบ่อย จึงจำเป็นต้องมีการผูกค่าใหม่อยู่บ่อยมาก ยกตัวอย่างเช่น เซตของโหนด R ที่เวลา t1 อาจจะแตกต่างอย่างสิ้นเชิงกับเซตของโหนด R ที่เวลา t2

```
Resource R = <expression1>
Time t1 = get_time();
x=Count(R);
     ...
Time t2 = get_time();
y=Count(R);
/* Normally, x != y */
```

จากตัวอย่างข้างต้นจะเห็นว่า เราเพียงแค่กำหนดเงื่อนไขเชิงประกาศสำหรับตัวแปรทรัพยากรเพื่อบรรยายทรัพยากรที่สนใจเท่านั้น แล้วระบบเราจะทำการบำรุงรักษาให้การผูกค่านั้นถูกต้องเองถึงแม้ว่าจะมีการเปลี่ยนแปลงเกิดขึ้นก็ตาม โดยทั่วไปแล้ว การอ้างอิงถึงตัวแปรทรัพยากรจะส่อความหมายโดยนัยถึงการเข้าถึงทรัพยากร ซึ่งการเข้าถึงตัวแปรทรัพยากรในระบบเรานี้มีความหมายที่ค่อนข้างเข้มงวด กล่าวคือ การเข้าถึงทรัพยากรจะต้องกระทำการบนทรัพยากรที่มีเงื่อนไขตรงตามที่ประกาศไว้ ณ เวลาที่ทำการเข้าถึง โดยที่เซตของโหนดในตัวแปรทรัพยากรจะต้องเปลี่ยนแปลงได้เองให้มีสมาชิกคือโหนดที่ตรงตามเงื่อนไขในขณะนั้นเท่านั้นได้โดยอัตโนมัติ โปรแกรมเมอร์ไม่ต้องเขียนโปรแกรมเพื่อจัดการในเรื่องนี้เลย ดังนั้น การจะรักษาไว้ซึ่งความหมายที่เข้มงวดนี้ ระบบจะต้องใช้พลังงานหรือโอเวอร์เฮดอย่างมีนัยสำคัญ เพื่อให้การผูกค่านี้ถูกต้องอยู่ตลอดเวลา ดังนั้น เราจึงเสนอทางเลือก หรือ พารามิเตอร์ปรับค่า ซึ่งสามารถใช้เพื่อประหยัดพลังงานในส่วนนี้โดยแลกกับความหมายของการเข้าถึงที่เข้มงวดน้อยลง ยกตัวอย่างเช่น โปรแกรมเมอร์สามารถลดความเข้มงวดได้โดยอนุญาตให้มีการเข้าถึงโหนดที่ผูกไว้ไม่เกิน t วินาทีก่อน

```
Resource R = <expression, last_bound_time > now-t>
```

นอกจากนี้ โปรแกรมเมอร์สามารถระบุงบพลังงานสำหรับการจำกัดการใช้พลังงานในการเข้าถึงทรัพยากรได้

```
Resource R = <expression, energy_budget = 100>
```

พารามิเตอร์ดังกล่าวเป็นเพียงแค่ตัวอย่างเท่านั้น อาจมีพารามิเตอร์อื่นๆอีกมากที่เป็นไปได้ ซึ่งเราจะละไว้สำหรับการวิจัยในภายหน้า

- การผูกแบบสถิต ถึงแม้การผูกแบบพลวัตจะดูสมเหตุสมผลดี อย่างไรก็ตามในบางสถานการณ์เราอาจจำเป็นต้องใช้การผูกแบบสถิต ยกตัวอย่างเช่น เราอาจต้องการเข้าถึงทรัพยากรที่ตรงตาม

เงื่อนไขมาก่อนซึ่งปัจจุบันทรัพยากรเหล่านั้นไม่มีคุณสมบัติตรงตามเงื่อนไขแล้ว เราอาจทำการเปิดกล้องในพื้นที่ A ให้ทำงานไว้ เมื่อเวลาผ่านไปเราอาจต้องการปิดมัน แต่ว่ามีกล้องบางตัวได้ถูกเคลื่อนย้ายไปบริเวณอื่นแล้ว ถ้าพื้นที่ A เป็นเงื่อนไขในการผูกทรัพยากรของเรา กล้องที่ถูกเคลื่อนย้ายไปพื้นที่อื่นจะไม่ตรงตามเงื่อนไขอีกต่อไป ทำให้เราไม่สามารถปิดกล้องเหล่านั้นได้โดยอ้างอิงถึงตัวแปรทรัพยากรเดิมได้

หนึ่งในวิธีแก้สำหรับปัญหาข้างต้นคือ การพึ่งพาระบบให้ช่วยจำคุณสมบัติของโหนดทุกตัวตั้งแต่เริ่มต้นทำงานจนถึงปัจจุบัน ยกตัวอย่างเช่น เราสามารถประกาศตัวแปรทรัพยากรตัวใหม่ที่มีเงื่อนไขเรื่องเวลามาประกอบด้วย ดังนี้

```
Resource R = <expression1>;
Time t1 = get_time();
    ....
Resource X = <expression1 && time == t1>;
```

ตราบใดก็ตามที่เรารู้เวลาที่โหนดตรงตามเงื่อนไข เรายอมสามารถอ้างอิงถึงมันอีกในอนาคตถึงแม้ว่ามันอาจไม่ตรงตามเงื่อนไขอีกแล้วได้ โดยการนำเวลาดังกล่าวมาผูกเป็นเงื่อนไขสำหรับตัวแปรทรัพยากรตัวใหม่สำหรับการเข้าถึงในภายหลังได้

วิธีแก้อีกวิธีที่คล้ายกันคือ การให้มีฟังก์ชัน last() ที่จะให้ค่าเซตที่แล้ว ที่ตรงตามเงื่อนไข ดังนั้น เราจะสามารถกระทำการบนเซตที่ต้องการได้ ถึงแม้ว่าในปัจจุบันเซตนั้นไม่ตรงตามเงื่อนไขอีกต่อไป เช่น

```
Resource R = <expression1>;
Resource X = last(R);
```

อย่างไรก็ตาม วิธีแก้ทั้งสองก่อให้เกิดโอเวอร์เฮดอย่างมาก เนื่องจากระบบจะต้องเก็บค่าการเปลี่ยนแปลงทุกอย่างของเซตไว้ตลอดเวลา

วิธีแก้ปัญหาอีกทางเลือกหนึ่งคือ การให้มีคำสั่งที่ระบุชัดเจนลงไปว่าให้จำเซตใดไว้ เราเสนอกลไกที่ระบุชัดเจนไว้อยู่ 2 กลไกคือ การใช้ตัวแปรทรัพยากรแบบสถิต และ การใช้ตัววนซ้ำ

ในการใช้ตัวแปรทรัพยากรแบบสถิตนั้น เราสามารถระบุว่าตัวแปรจะผูกกับค่าใดไว้โดยไม่เปลี่ยนแปลงไปตามเงื่อนไขอีกต่อไป จะไม่มีการผูกใหม่ไม่ว่าจะเกิดเหตุการณ์ใดขึ้นก็ตาม ดังนั้นเราสามารถเก็บเซตของโหนดใดใดไว้ได้ ถึงแม้ว่าในเวลาต่อมา มันจะไม่ตรงตามเงื่อนไขอีกแล้ว ดังตัวอย่าง

```
Resource R1;
Static Resource R2=R1;
/* R1 changes over time but R2 does not*/
```

คำสั่งที่ระบุชัดเจนเช่นนี้ จะมีโอเวอร์เฮดน้อยกว่า last() อย่างมากเพราะว่าระบบไม่ต้องเก็บค่าทุกค่าที่เคยตรงตามเงื่อนไขมาก่อนของทุกตัวแปรทรัพยากร ยิ่งกว่านั้น ตัวแปรทรัพยากรแบบสถิตนี้เราตั้งใจให้มันจดจำเซตของโหนดที่ตรงตามเงื่อนไข แต่หากเราต้องการจะจดจำเพียงแค่โหนดเดียว การใช้ตัววนซ้ำจะเหมาะสมกว่า โดยค่าของตัววนซ้ำจะไม่เปลี่ยนแปลงแบบอัตโนมัติโดยปราศจากการหมอบหมายค่าใหม่ให้ ดังตัวอย่างต่อไปนี้

```
Iterator i1 = R1->first_element;
```

2.7 เวลาสิ้นสุดการเข้าถึง

ไม่ว่าเราจะใช้การผูกค่าตัวแปรทรัพยากรแบบใดก็ตาม การเข้าถึงทรัพยากรจะสำเร็จหรือไม่ เป็นเรื่องที่ไม่สามารถรับประกันได้ร้อยเปอร์เซ็นต์ เวลาในการเข้าถึงทรัพยากรในเครือข่ายเซ็นเซอร์ไร้สายนั้นไม่สามารถกำหนดขอบเขตได้ การล้มเหลวของการเข้าถึงทรัพยากรเป็นเรื่องที่ไม่อาจหลีกเลี่ยงได้ในทางปฏิบัติ เนื่องจากสภาพที่เป็นพลวัตของเครือข่าย ดังนั้น เราไม่สามารถแยกแยะความแตกต่างระหว่างเหตุการณ์สองเหตุการณ์นี้ได้โดยง่าย เหตุการณ์สองเหตุการณ์ดังกล่าวนั้น คือ เหตุการณ์แรกเป็นเหตุการณ์ที่การเข้าถึงนั้นสำเร็จแต่ไม่สามารถกำหนดขอบเขตว่าใช้เวลาเท่าใด หากเรารอนานพอจะสามารถเข้าถึงได้ แต่อาจจะต้องรอนานมากแล้วไม่รู้ว่าจะต้องรอนานเท่าใด ส่วนเหตุการณ์ที่สองเป็นเหตุการณ์ที่การเข้าถึงนั้นล้มเหลวไม่ว่าจะรอนานเท่าใดก็ตาม

ในที่นี้ เราขอเสนอให้ทำการสนธิตัวแปรทรัพยากรเข้ากับเวลาสิ้นสุดในการเข้าถึง หากการเข้าถึงใด ในเวลานานกว่าเวลาสิ้นสุดของการเข้าถึง ถือว่าเกิดสิ่งผิดปกติขึ้น อันนำไปสู่ Exception (ในลักษณะคล้ายกับ Java Exception) ในที่นี้โปรแกรมเมอร์ควรจะต้องสามารถระบุหรือเขียนคำสั่งเพื่อจัดการกับปัญหานี้อย่างชัดเจนได้ โดยใช้ catch statement ดังตัวอย่างต่อไปนี้

```
Resource R = <expression1, timeout = 10>
Iterator i = R->first_element;
try {
     printf("light intensity = %f", i->light);
} catch(TimeoutException) {
printf("can't access the light sensor");
}
```