جَـامعة
المَـنارة
MANARA UNIVERSITY

محاضرة2 د.فادي متوج

MODELS OF COMPUTATION AND MODELING LANGUAGES

1. System Specification
2. System Modeling and Formal Models
3. Models of Computation: What's that?
4. Concurrency
5. Communication & Synchronisation
6. Common Models of Computation

محاور المحاضرة

1. مواصفات النظام
   - كيف يمكننا وصف ما يجب على النظام أن يفعله بشكل دقيق لا يقبل اللبس، قبل أن نبدأ حتى في التفكير بكيفية بنائه؟
2. نمذجة النظام والنماذج الرسمية
   - سنتعلم كيف نترجم هذه "المواصفات" إلى "نماذج ". وسنركز على أهمية النماذج الرسمية (Formal Models)، وهي تلك التي لها أساس رياضي صارم، لأنها الوحيدة التي يمكننا تحليلها والتحقق من صحتها بثقة تامة.
3. نماذج الحوسبة ما هي؟
   - "نموذج الحوسبة" هو مجموعة من القواعد التي تحدد "قوانين الفيزياء" للعالم الافتراضي لنظامنا. هو يحدد كيف تتفاعل المكونات، وكيف يمر الزمن، وما هو المسموح وما هو الممنوع..
4. التزامن

- كيف تتعامل نماذج الحوسبة المختلفة مع فكرة حدوث عدة أشياء في نفس الوقت؟ هذا هو أحد الفروقات الأساسية بين النماذج.

5. الاتصال والمزامنة

- إذا كانت هناك أشياء تحدث بالتوازي، فكيف تتواصل مع بعضها البعض؟ وكيف تنسق أعمالها؟ سندرس الآليات المختلفة التي توفرها النماذج المختلفة للتواصل والتنسيق.

6. نماذج الحوسبة الشائعة

- أخيراً، سنقوم بجولة على أشهر وأهم نماذج الحوسبة المستخدمة اليوم في تصميم الأنظمة المدمجة .

From Specifications to Implementations

- **Specification: A description of basic requirements and properties of a system**

  - **The designer gets a *specification* as an input and, finally, has to produce an *implementation*.**
    This is usually done as a sequence of *refinement steps*.

  - **Specifications can be:**
    - **informal (natural language)**
    - **more detailed and unambiguous (based on a formal notation)**

1. ما هي المواصفات؟

ببساطة، المواصفات هي وثيقة "ماذا نريد؟". إنها الوصف الدقيق لما يجب أن يفعله النظام، وما هي خصائصه، وما هي حدوده وقيوده.

مثال :

لنفترض أن شركة سيارات طلبت تصميم روبوت صغير ذاتي القيادة (Autonomous Mobile Robot) لنقل قطع الغيار في مصنعها.

- الفكرة الأولية (غير كافية) :" نريد روبوت ينقل القطع"
- المواصفات (Specification) :هي الإجابة التفصيلية على أسئلة مثل:
  - ما هو الحمل الأقصى الذي يجب أن يحمله الروبوت؟ (مثلاً: 20 كيلوغرام).

- ما هي السرعة القصوى للروبوت؟ (مثلاً: 1 متر في الثانية).
- كم ساعة يجب أن تعمل البطارية قبل إعادة الشحن؟ (مثلاً: 8 ساعات عمل متواصل).
- ما مدى الدقة التي يجب أن يتوقف بها عند المحطات؟ (مثلاً: بدقة ±2 سنتيمتر).
- هل يجب أن يستطيع تجنب العوائق المتحركة والثابتة؟ (نعم، باستخدام حساسات LiDAR وكاميرات).

هذه القائمة هي "المواصفات". بدونها، سيكون عملنا كمهندسين عشوائياً وغير فعال.

---

2. المصمم يأخذ المواصفات كدخل(input) ، وفي النهاية، يجب أن ينتج تنفيذاً .

- المواصفات (Input) : هي قائمة "ماذا نريد؟" التي ذكرناها في المثال.
- التنفيذ (Output) : هو المنتج النهائي الفعلي. هو الروبوت الحقيقي الذي صنعناه، بكل مكوناته المادية (Hardware)وبرامجه (Software) التي تحقق كل نقطة في قائمة المواصفات.
- خطوات التحسين (Refinement Steps) : نحن لا نقفز من قائمة المتطلبات إلى الروبوت الجاهز مرة واحدة بل نمر بمراحل:

1. تصميم عالي المستوى :اختيار المعالج الصغري(Microcontroller) ، نوع المحركات، نوع الحساسات.
2. تصميم تفصيلي :تصميم الدارات الإلكترونية، كتابة الخوارزميات البرمجية للملاحة وتجنب العقبات.
3. بناء نموذج أولي(Prototype)
4. الاختبار والتعديل .كل خطوة من هذه هي"Refinement Step" ، حيث تضيف تفاصيل أكثر وتجعل التصميم أقرب إلى الواقع.

---

3. أنواع المواصفات

يمكن أن تأتي المواصفات بشكلين:

✓ مواصفات غير رسمية(Informal)

تكون مكتوبة بلغة البشر العادية(natural language)

- مثال :مدير المصنع يقول لك: "أريدك أن تصنع لي روبوتاً ذكياً وسريعاً لنقل الصناديق من النقطة A إلى النقطة B بأمان"
- مشكلتها :هذه اللغة غامضة (ambiguous) ما معنى "ذكي"؟ ما هو تعريف "سريع"؟ ما المقصود بـ"بأمان"؟ هذه الكلمات تحتمل أكثر من معنى وتسبب سوء فهم بين العميل والمهندس.

✓ مواصفات رسمية ومفصلة(Formal)

تكون مكتوبة بلغة دقيقة، غالباً باستخدام الرياضيات، الأرقام، والرموز المحددة التي لا تحتمل إلا معنى واحد.

- مثال (ترجمة للمثال غير الرسمي):
  - يجب على الروبوت نقل حمولة بوزن 20 كغ من الإحداثيات (x1, y1) إلى(x2, y2)
  - يجب أن يقطع المسافة في زمن أقل من 60 ثانية(t < 60s)
  - يجب أن يتوقف النظام تلقائياً إذا اقترب أي جسم من مساره على بعد أقل من 50 سم

هذا النوع من المواصفات هو ما يفضله المهندسون لأنه لا يترك مجالاً للشك أو سوء الفهم.

الخلاصة :إن أول خطوة وأهمها في بناء أي نظام مدمج، من ساعة يد ذكية إلى ذراع روبوتية معقدة، هي تحويل فكرة العميل الغامضة إلى قائمة مواصفات دقيقة وواضحة. كلما كانت المواصفات مفصلة، كان المنتج النهائي أفضل وأقرب لما هو مطلوب.

System Specifications

- **A specification captures:**
  - **The basic required behaviour of the system**
    - **E.g. as a relation between inputs and outputs**
  - **Other (non-functional) requirements**
    - **time constraints**
    - **power/energy constraints**
    - **safety requirements**
    - **environmental aspects**
    - **cost, weight, etc.**

إن وثيقة المواصفات تحتوي على نوعين رئيسيين من المعلومات:

1. **المتطلبات الوظيفية (Functional Requirements)** : ماذا يفعل النظام؟
2. **المتطلبات غير الوظيفية (Non-Functional Requirements)** :كيف يجب أن يفعل ذلك؟

---

1. السلوك الأساسي المطلوب (المتطلبات الوظيفية)

هذا هو قلب النظام، وظيفته الأساسية. **" العلاقة بين المدخلات والمخرجات "**

يمكن أن نفكر في أي نظام مدمج على أنه صندوق أسود: نعطيه شيئاً (مدخلات)، فيقوم هو بإعطائنا شيئاً آخر (مخرجات). المتطلبات الوظيفية تصف هذه العلاقة بدقة.

مثال : لنكمل مع روبوت المصنع ذاتي القيادة

- **مثال وظيفي 1:**
  - **المدخل (Input)** : الروبوت يستقبل أمراً لاسلكياً بالذهاب إلى محطة الشحن رقم 3
  - **المخرج (Output)** : الروبوت يحدد مساره بنفسه ويتحرك إلى إحداثيات المحطة رقم 3
- **مثال وظيفي 2:**
  - **المدخل (Input)** : حساس الأجسام (LiDAR) يكتشف وجود عامل يقف في مسار الروبوت على بعد متر واحد.
  - **المخرج (Output)** : نظام التحكم في المحركات يوقف دوران العجلات فوراً.

ببساطة، المتطلبات الوظيفية هي قائمة المهام التي يجب على الروبوت تنفيذها.

---

2. المتطلبات الأخرى (غير الوظيفية)

هذه المتطلبات لا تصف "ماذا" يفعله النظام، بل تصف **القيود والشروط** التي يجب أن يعمل ضمنها. هي التي تحدد جودة النظام وتجعله ناجحاً أو فاشلاً في الواقع العملي. في عالم الأنظمة المدمجة، هذه المتطلبات قد تكون أهم من المتطلبات الوظيفية.

لنأخذ قائمة من هذه المتطلبات ونطبقها على مثال الروبوت:

- **قيود الزمن(Time constraints) :**
  - عندما يكتشف الروبوت عاملاً في مساره، يجب أن يتوقف **خلال 150 ميللي ثانية**.
  - هذا القيد الزمني هو الفارق بين نظام آمن وكارثة في المصنع.
- **قيود الطاقة(Power constraints) :**
  - يجب أن يستهلك الروبوت طاقة بمعدل لا يزيد عن 40 واط أثناء حركته، لضمان عمل البطارية **لمدة 10 ساعات**.
  - لا فائدة من روبوت يؤدي وظيفته لكن بطاريته تنفد كل ساعة.
- **متطلبات الأمان(Safety requirements) :**

- يجب أن يحتوي الروبوت على زر **إيقاف طوارئ (Emergency Stop)** أحمر وواضح. عند الضغط عليه، يجب أن تتوقف جميع المحركات فوراً وبشكل فيزيائي (وليس فقط برمجياً).

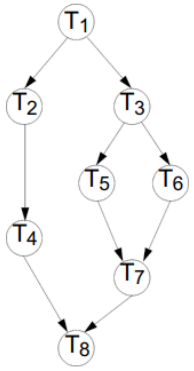
- **الجوانب البيئية(Environmental aspects) :**
  - يجب أن يعمل الروبوت بكفاءة في درجات حرارة تتراوح بين **0و50 درجة مئوية** ورطوبة تصل إلى 85%
  - هذا يضمن أن الروبوت لن يتعطل في صيف حار أو شتاء بارد داخل المصنع.

- **التكلفة والوزن(Cost, Weight) :**
  - يجب ألا تتجاوز **تكلفة** تصنيع الروبوت الواحد 3000 دولار .
  - يجب ألا يتجاوز **وزن** الروبوت مع بطاريته 40 كيلوغراماً ليسهل نقله وصيانته.

System Model

- As an early step in the design flow, a *system model* is produced (you remember the design flow!).
- The model is a description of certain aspects/properties of the system. Models are abstract, in the sense that they omit details and concentrate on aspects that are significant for the design process.

You remember our task graph example!



ببساطة، بعد أن نحصل على قائمة المواصفات، لا نبدأ مباشرة في كتابة الكود أو تصميم الدارات الإلكترونية. هذا سيكون مثل محاولة بناء ناطحة سحاب بدون مخططات هندسية. بدلاً من ذلك، نقوم بإنشاء "نموذج".

---

1. ما هو نموذج النظام؟

النموذج هو **رسم تخطيطي مبسط** للنظام. يمكن أن نفكر فيه كخريطة. خريطة المدينة لا تظهر لنا كل شجرة وكل سيارة وكل شخص. لأن ذلك سيجعلها عديمة الفائدة. الخريطة تتجاهل التفاصيل غير المهمة (omits details) وتركز على الأشياء الأساسية التي نحتاجها: الشوارع الرئيسية، الأماكن الهامة، وهكذا.

هذا بالضبط ما يفعله نموذج النظام. إنه **تجريد (abstract)** للواقع. نحن نتجاهل مؤقتاً التفاصيل المعقدة (مثل نوع المعالج، أو لغة البرمجة) ونركز على جانب واحد مهم من النظام لنفهمه جيداً.

---

2. مثال عملي: مخطط المهام (Task Graph)

يمثل المخطط مثال رائع على نموذج. هذا المخطط يسمى "Task Graph" . سوف نطبقه على مثال روبوتي لنفهم قوته.

لنتخيل أن لدينا ذراعاً روبوتية وظيفتها هي أن **ترى** جسماً، ثم **تلتقطه** وتضعه في مكان آخر. هذه العملية تتكون من عدة مهام برمجية. هذا النموذج (المخطط) يوضح **ترتيب واعتمادية** هذه المهام.

لنفترض أن الدوائر في المخطط (T1, T2, etc.) تمثل المهام التالية:

- **T1: إلتقاط صورة(Capture Image)**
  - هذه هي المهمة الأولى دائماً.
- **T2: معالجة الصورة وتحديد مكان الجسم(Process Image)**
  - هذه المهمة لا يمكن أن تبدأ إلا **بعد** أن تكتمل T1 (بعد التقاط الصورة). السهم من T1 إلى T2 يمثل هذه الاعتمادية.
- **T3: فحص حساسات الأمان(Check Safety Sensors)**
  - هذه المهمة تتأكد من عدم وجود أي عائق (مثل يد عامل) في منطقة عمل الروبوت. يمكن أن تبدأ أيضاً **بعد** T1
- **T4: حساب مسار حركة الذراع(Calculate Arm Path)**
  - لا يمكن حساب المسار الصحيح إلا **بعد** معرفة مكان الجسم. لذلك، T4 تعتمد علىT2
- **T5و T6 : مهام فحص وتشخيص أخرى(Diagnostics)**
  - تعتمد على أن حساسات الأمان تعمل (تعتمد علىT3).
- **T7: تنفيذ الحركة والالتقاط(Move Arm & Grip)**
  - هذه هي المهمة الحرجة. لا يمكن للذراع أن تتحرك وتلتقط الجسم إلا **بعد** حساب المسار (T4) **وبعد** التأكد من أن كل فحوصات الأمان والتشخيص سليمة(T5 و T6) نلاحظ كيف أن ثلاثة أسهم تصب فيT7

- **T8: إرسال إشارة "اكتملت المهمة(Signal Completion) "**
  - لا يمكن إرسال هذه الإشارة إلا **بعد** أن تنتهي حركة الذراعT7

ما الذي تعلمناه من هذا النموذج؟

1. **الاعتماديات :**فهمنا بوضوح أي مهمة تعتمد على الأخرى. لا يمكننا أن نطلب من الروبوت أن يحسب مسار الحركة (T4) قبل أن يحدد مكان الجسم(T2)
2. **التوازي (Parallelism)** : لاحظنا أن المهمتين T2 و T3 لا تعتمدان على بعضهما. هذا يعني أنه إذا كان لدينا معالج متعدد الأنوية(multi-core processor) ، يمكننا تشغيل هاتين المهمتين **في نفس الوقت** على نواتين مختلفتين لتسريع العملية كلها. هذا قرار تصميمي مهم جداً، اكتشفناه من خلال النموذج قبل كتابة أي سطر برمجي.

الخلاصة :النموذج هو أداة قوية للمهندس. إنه يبسط الواقع، ويساعدنا على تحليل النظام، ورؤية المشاكل المحتملة، واتخاذ قرارات تصميم ذكية في مرحلة مبكرة جداً من المشروع، مما يوفر الكثير من الوقت والمال.

System Model

- Models are formulated using *modeling languages*
- Modeling language:
  - well-suited to expressing the basic system properties and basic aspects of system behaviour in a succinct and clear manner
  - lends itself well to the, preferably automatic, checking of requirements and synthesis of implementations.
- Depending on the particularities of the system, an adequate modeling language has to be chosen.
  The language has to contain the appropriate language constructs in order to express the system's functionality and requirements.

بعد أن فهمنا "لماذا" نستخدم النماذج، سنجيب على سؤال "كيف" نبني هذه النماذج. الجواب هو: باستخدام **"لغات النمذجة(Modeling Languages) ".**

---

1. ما هي "لغة النمذجة"؟

عندما نتحدث عن "لغة"، قد يتبادر إلى ذهننا C++ أو Python ، لكن لغة النمذجة شيء مختلف. إنها ليست لغة برمجة نستخدمها لكتابة المنتج النهائي، بل هي **مجموعة من الرموزوالقواعد الموحدة** نستخدمها لوصف ورسم النموذج.

أفضل مثال هو المثال السابق :مخطط المهام (Task Graph) هو بحد ذاته لغة نمذجة بصرية.

- **مفردات اللغة (Symbols)** : الدوائر (التي تمثل المهام) والأسهم (التي تمثل الاعتماديات).
- **قواعد اللغة (Rules)** : السهم يجب أن يخرج من مهمة ويدخل في أخرى، ولا يمكن لمهمة أن تبدأ قبل انتهاء جميع المهام التي تشير إليها الأسهم.

باستخدام هذه اللغة البسيطة، استطعنا وصف نظام معقد.

---

2. صفات لغة النمذجة الجيدة

ليست كل لغات النمذجة متساوية. الشريحة يمكن تحديد صفتين للغة الجيدة:

أ) موجزة وواضحة(Succinct and Clear)

يجب أن تسمح لنا اللغة بالتعبير عن سلوك وخصائص النظام بطريقة مختصرة وواضحة. مخطط المهام السابق عبر عن علاقات معقدة بين 8 مهام في رسم بسيط، بينما كتابة نفس الوصف باللغة العادية سيستغرق فقرة طويلة ومربكة.

ب) تسمح بالفحص التلقائي والتوليف(Automatic Checking and Synthesis)

هذه هي القوة الحقيقية للنمذجة الحديثة.

- **الفحص التلقائي (Automatic Checking)** : يمكننا استخدام برامج الكمبيوتر لتحليل النموذج الذي رسمناه **والبحث عن أخطاء منطقية** قبل كتابة أي كود.
  - **مثال :**لنتخيل أننا في مخطط المهام رسمنا بالخطأ سهماً من T4 إلى T1 هذا سيخلق حلقة مغلقة مستحيلة(Deadlock) ، حيث T4 تنتظر T2 ، و T2 تنتظر T1 التي بدورها تنتظر .T4 برنامج التحليل سيعطينا تحذير "لديك حلقة مميتة في تصميمك!". اكتشاف هذا الخطأ الآن يوفر أياماً من البحث عن سبب تجمد الروبوت لاحقاً.
- **التوليف (Synthesis)** : هذا يعني أن برامج الكمبيوتر يمكنها **توليد الكود البرمجي أو أجزاء من تصميم الهاردوير تلقائياً** بناءً على النموذج.
  - **مثال :** يمكننا رسم سلوك الروبوت كنموذج يسمى "مخطط الحالات (State Machine Diagram) " يصف حالاته المختلفة (خامل، يتحرك، يشحن، طوارئ) . بعدها، يمكن لأداة برمجية أن تأخذ هذا الرسم وتولد تلقائياً الهيكل الأساسي لكود C++ مثلاً (switch-case statement) الذي ينفذ هذا السلوك. هذا يقلل الأخطاء البشرية ويسرع التطوير بشكل هائل.

3. **اختيار اللغة المناسبة للعمل المناسب**

لا توجد لغة نمذجة واحدة هي الأفضل لكل شيء. يجب على المهندس اختيار لغة النمذجة المناسبة للمشكلة التي يريد حلها.

أمثلة من عالم الروبوتات:

- **لنمذجة تدفق البيانات والمهام** :نستخدم لغات مثل (**Dataflow Graphs** أو **Task Graphs**) مثل الذي رأيناه.
- **لنمذجة سلوك النظام المتغير** : نستخدم لغات مثل **State Machines** أو **State charts** هذا مثالي لوصف منطق التحكم في الروبوت.
- **لنمذجة التوقيت والقيود الزمنية** :نستخدم لغات متخصصة بتحليل التوقيت(Timing analysis)
- **لنمذجة الأنظمة الفيزيائية (حركة الذراع، المحركات)** :نستخدم لغات رياضية وأدوات محاكاة مثل **Simulink** في برنامجMATLAB

الخلاصة :لغات النمذجة هي أدواتنا الرسمية لوصف وتصميم الأنظمة. اللغة الجيدة تجعل التصميم واضحاً وتسمح للكمبيوتر بمساعدتنا في إيجاد الأخطاء وكتابة الكود. و يجب علينا كمهندسين أن نعرف أي أداة (لغة) نستخدمها لأي جزء من تصميم الروبوت.

System Model

- **Modeling Languages can be**
  - **graphical**
  - **textual**

- **Modeling languages can be**
  - **“ordinary” programming languages (C, C++)**
  - **hardware description languages (VHDL, Verilog)**
  - **languages specialised for modeling of systems in particular areas, and with particular features;**
    they are often based on particular *models of computation*.

يمكن تصنيف لغات النمذجة بطريقتين بسيطتين:

1. حسب شكلها :كيف تبدو اللغة؟

2. حسب نوعها :لأي غرض تُستخدم اللغة؟

لنتناول كل تصنيف على حدة.

1. التصنيف حسب الشكل: رسومية أم نصية؟

أ) لغات رسومية (Graphical)

هذه هي اللغات التي "نرسمها". نستخدم فيها الأشكال والرموز والخطوط والأسهم لوصف النظام.

- المثال الذي نعرفه :مخطط المهام (Task Graph) الذي رأيناه قبل قليل هو لغة رسومية. بمجرد النظر إليه، فهمنا الفكرة العامة.
- مثال آخر مشهور في الروبوتات: " مخططات الحالة(State Machine Diagrams) "، حيث نرسم دوائر لتمثل حالات الروبوت (مثلاً :واقف، يتحرك للأمام، يتجنب عقبة) وأسهم لتمثل الانتقال بين هذه الحالات.
- ميزتها :سهلة الفهم بصرياً، ورائعة للتواصل بين أعضاء الفريق وشرح الأفكار المعقدة بسرعة.

ب) لغات نصية (Textual)

هذه اللغات تشبه لغات البرمجة، حيث نكتب سطوراً من النص لوصف سلوك أو هيكل النظام.

- مثال :بدلاً من رسم مخطط الحالة، يمكننا كتابة وصف نصي له، كالتالي:

```
state(IDLE) {
 on(start_command) -> transition_to(MOVING);
}
state(MOVING) {
 on(obstacle_detected) -> transition_to(AVOIDING);
}
```

- ميزتها :يمكن أن تكون أكثر دقة وتفصيلاً. من السهل جداً تتبع التغييرات فيها باستخدام أنظمة مثل"Git" ، ومن السهل على الكمبيوتر تحليلها ومعالجتها.

2. التصنيف حسب النوع والاستخدام

هنا نتعمق أكثر في أنواع اللغات التي نستخدمها كمهندسين.

أ) لغات البرمجة العادية مثل(C, C++).

قلنا سابقاً أن لغات النمذجة ليست لغات برمجة، لكن يمكننا استخدام لغة برمجة عادية مثل C++ كأداة لبناء نموذج محاكاة (Simulation Model) للنظام.

- مثال :قبل أن نكتب الكود النهائي الذي سيعمل على المعالج المحدود للروبوت، يمكننا كتابة برنامج C++ على حاسوبنا. هذا البرنامج يقلد (يحاكي) سلوك الروبوت وحساساته. يمكننا أن نختبر خوارزميات الملاحة وتجنب العقبات على هذا النموذج البرمجي آلاف المرات في دقائق، وهو أمر مستحيل على الروبوت الحقيقي. هنا، برنامج الـ C++ هو "النموذج".

ب) لغات وصف العتاد(VHDL, Verilog)

هذه لغات نصية متخصصة جداً. وظيفتها ليست وصف البرامج، بل وصف وتصميم الدارات الرقمية الإلكترونية مثل المعالجات والشرائح المتخصصة.(FPGAs, ASICs)

- مثال : لنفترض أن روبوت يحتاج إلى معالجة الفيديو من الكاميرا بسرعة فائقة لتتبع جسم متحرك. قد يكون المعالج العادي بطيئاً. لذا، قررنا تصميم شريحة إلكترونية (FPGA) مخصصة لهذه المهمة فقط. سنستخدم لغة مثل VHDL لوصف كل بوابة منطقية وكل مسجل في هذه الشريحة. هذا الوصف هو "نموذج" كامل للعتاد (Hardware) قبل تصنيعه.

ج) اللغات المتخصصة(Specialized Languages)

هذه هي لغات النمذجة "الحقيقية" التي صممت خصيصاً لهدف النمذجة في مجال معين. إنها غالباً ما تكون مبنية على "نماذج للحوسبة (Models of Computation) ". ببساطة تعني أنها مبنية على "طريقة تفكير" معينة في كيفية عمل النظام.

- مثال : لغة Simulink في MATLAB .هذه لغة رسومية مبنية على نموذج حوسبة اسمه "تدفق البيانات " (Dataflow). حيث نبني نموذجنا على شكل صناديق(Blocks) ، والبيانات "تتدفق" من صندوق لآخر. هذا مثالي لنمذجة أنظمة التحكم في الروبوت، حيث تتدفق قراءات الحساسات إلى فلاتر، ثم إلى متحكمات، ثم إلى أوامر للمحركات.

الخلاصة :لدينا صندوق أدوات كبير من لغات النمذجة. بعضها رسومي وبديهي، وبعضها نصي ودقيق. بعضها لغات عامة نستخدمها للمحاكاة(C++) ، وبعضها شديد التخصص لتصميم الهاردوير (VHDL) أو أنظمة التحكم. (Simulink) اختيار الأداة الصحيحة هو من أهم مهارات مهندس الأنظمة المدمجة.

System Model

What do we want to do with the model of an embedded system?

1. **To validate the system description in order to check that the specified functionality is the desired one and the requirements are stated correctly:**
   - **by formal verification**
   - **by simulation**
2. **To synthesise efficient implementations**

هنا سنجيب على سؤال جوهري: " لقد تعبنا وبنينا هذا النموذج ...والآن ماذا؟ "ما الفائدة العملية الحقيقية منه؟

هناك هدفين رئيسيين وقويين جداً لاستخدام النماذج في تصميم الأنظمة المدمجة:

1. التحقق والتأكد (Validation)

الهدف الأول هو التأكد من أننا نبني النظام الصحيح. قبل أن ننفق الكثير من الوقت والمال في بناء الروبوت، نستخدم النموذج لنتأكد من أن تصميمنا يفي بالمتطلبات وأننا فهمنا ما هو مطلوب منا بشكل صحيح. هذا يشبه قيام مهندس معماري بصنع مجسم ثلاثي الأبعاد للمبنى للعميل ليتأكد من أن كل شيء في مكانه الصحيح قبل البدء في الحفر.

توجد هناك طريقتين للتحقق:

أ) الإثبات الرسمي (Formal Verification)

هذه هي الطريقة الأكثر صرامة وقوة. إنها تستخدم الرياضيات والمنطق الحاسوبي لتُثبت بشكل قاطع أن تصميمنا (النموذج) لن ينتهك أبداً قاعدة معينة ومهمة.

- مثال : لدينا متطلب أمان حرج يقول: " يجب ألا تصل سرعة الروبوت أبداً إلى الصفر في منتصف تقاطع طرق داخل المصنع (لتجنب عرقلة الحركة)". يمكننا استخدام أداة إثبات رسمي لتحليل نموذج التحكم الخاص بنا. ستقوم الأداة باستكشاف كل السيناريوهات الممكنة رياضياً. إذا وجدت أي احتمال، مهما كان نادراً، يمكن أن يؤدي إلى توقف الروبوت في التقاطع، فستنبهنا. وإن لم تجد، فهي بذلك قد أثبتت رياضياً أن هذا الخطأ مستحيل الحدوث. هذا أقوى بكثير من مجرد الاختبار العادي.

ب) المحاكاة (Simulation)

المحاكاة هي "قيادة تجريبية" افتراضية لنموذجنا. نحن نضع النموذج في بيئة افتراضية ونرى كيف يتصرف.

- مثال : نصنع نموذجاً لروبوت التوصيل الخاص بنا، ثم نضعه داخل بيئة محاكاة ثلاثية الأبعاد للمدينة التي سيعمل بها .يمكننا بعد ذلك أن نختبر سيناريوهات مختلفة: ماذا يحدث لو ظهر طفل فجأة أمامه؟ ماذا لو كانت إشارة المرور حمراء؟ ماذا لو انقطع الاتصال بالإنترنت؟ من خلال مراقبة سلوك النموذج في المحاكاة، يمكننا التحقق من أن وظائفه تعمل كما هو متوقع في ظروف شبه حقيقية، وكل ذلك ونحن بأمان في مكاتبنا.

2. التوليف (Synthesis)

الهدف الثاني هو استخدام النموذج لتوليد (بناء) التنفيذ الفعلي بشكل تلقائي .هنا، لا يكون النموذج مجرد رسم للتوضيح، بل يصبح مخططاً تقنياً يمكن للكمبيوتر قراءته واستخدامه لبناء النظام الحقيقي.

- مثال (توليف البرمجيات) :بعد أن نصمم "مخطط الحالة (State Machine) "الذي يصف سلوك الروبوت، يمكننا استخدام أداة برمجية (Code Generator) تقوم بقراءة هذا المخطط الرسومي وتوليد 80% من كود C++ تلقائياً .هذا يوفر وقتاً هائلاً ويقلل من الأخطاء البشرية في كتابة الكود.
- مثال (توليف العتاد) :كما ذكرنا سابقاً، عندما يكتب مهندس الإلكترونيات وصفاً لشريحة بمعالج الصور بلغة VHDL(وهو نموذج نصي)، فإنه يستخدم أداة "توليف" تأخذ هذا الوصف وتحوله تلقائياً إلى خريطة التوصيلات الفعلية للبوابات المنطقية على شريحة الـFPGA

الخلاصة :النموذج ليس مجرد خطوة نظرية، بل هو أداة عملية حيوية:

1. أولاً، نستخدمه لنتأكد من أن تصميمنا صحيح وآمن(Validation)
2. ثانياً، نستخدمه كنقطة انطلاق لبناء النظام الفعلي بشكل أسرع وأكثر دقة(Synthesis)

Semantics of System Models

We would like modeling languages to have well defined semantics ⇒ models are unambiguous.

- **The *semantics* is the set of rules which associate a meaning to *syntactical* constructs (combination of symbols) of the language.**
- **The semantics of the language is based on the underlying *model of computation*.**

  It depends on this underlying model of computation what kind of systems can be described with the language.

  The model of computation decides on the *expressiveness* of the language.

- دلالات نماذج الأنظمة

- يجب أن تكون لغات النمذجة ذات دلالات محددة جيداً لتكون النماذج غير غامضة. الدلالات هي القواعد التي تربط معنى بالتركيبات اللغوية، وتعتمد على نموذج الحساب الأساسي الذي يحدد قدرة اللغة على التعبير.
- **مثال :**أمر "تحرك للأمام" في لغة نمذجة الروبوت يجب أن يكون له معنى واحد وواضح لتجنب سلوك غير متوقع. يوفر نموذج الحساب الأدوات الأساسية لوصف سلوك النظام ويؤثر على مدى تعقيد الأنظمة التي يمكن وصفها.

دلالات نماذج الأنظمة هي نقطة أساسية جداً في فهم لغات النمذجة التي نستخدمها لتصميم الأنظمة المدمجة، بما في ذلك تلك التي نستخدمها في الروبوتات.

1- نريد أن تكون لغات النمذجة ذات دلالات محددة جيداً، وهذا يعني أن النماذج يجب أن تكون غيرغامضة.

لنتخيل أننا نعطي أمراً لروبوت ما، ولكن هذا الأمر يمكن تفسيره بطريقتين مختلفتين. هذا سيؤدي إلى سلوك غير متوقع، وقد يكون خطيراً في بعض الأحيان. نفس الشيء ينطبق على لغات النمذجة التي نستخدمها لوصف سلوك الأنظمة المدمجة. يجب أن تكون هذه اللغات واضحة بحيث يفهم المصمم والجهاز (الذي سيبنى بناءً على هذا النموذج) نفس الشيء بالضبط.

مثال: لنفترض أن لدينا أمر في لغة نمذجة الروبوت يقول: "تحرك للأمام". إذا لم تكن دلالات هذه اللغة محددة جيداً، فقد يفهم الروبوت "تحرك للأمام بسرعة بطيئة" بينما يقصد المصمم "تحرك للأمام بسرعة عالية". هذا الغموض قد يؤدي إلى مشاكل في أداء الروبوت.

2- الدلالات هي مجموعة القواعد التي تربط معنى بالتركيبات اللغوية (مجموعات الرموز) للغة.

يمكن التفكير في الأمر كقواعد اللغة العربية أو الإنجليزية. لكل كلمة ولكل طريقة لتركيب الكلمات معاً معنى محدد.
الدلالات في لغات النمذجة هي نفسها، لكنها تحدد معنى العناصر والتركيبات المستخدمة في وصف النظام المدمج.
مثال: في لغة نمذجة قد نستخدم رمزاً معيناً (مثل مربع بداخله حرف 'S') لتمثيل "حالة البدء" في نظام معين. الدلالات المحددة جيداً لهذه اللغة ستوضح أن هذا الرمز يعني بالضبط النقطة التي يبدأ منها تنفيذ النظام.
3- تعتمد دلالات اللغة على نموذج الحساب الأساسي، وهذا النموذج يحدد أنواع الأنظمة التي يمكن وصفها باللغة، وبالتالي يحدد مدى قدرة اللغة على التعبير.

Semantics of System Models

Do we want large expressiveness (we can describe anything we want)?
Not exactly!

- **Large expressive power: imperative model (e.g. unrestricted use of C or Java):**
  - **Can specify "anything".**
  - **No formal reasoning possible (or extremely complex).**

- **Limited expressive power, based on well chosen computation model:**
  - **Only particular systems can be specified.**
  - **Formal reasoning is possible.**
  - **Efficient (possibly automatic) synthesis.**

هل نريد لغة نمذجة "خارقة" تستطيع أن تصف أي شيء نريده؟
الجواب المفاجئ، هو :**لا، ليس تماماً**
هناك مقايضة أساسية (trade-off) بين قوة اللغة وقابليتها للتحليل. سوف نوضح ذلك.

---

1. لغات ذات قوة تعبيرية هائلة (مثل C/C++)

هذه هي اللغات التي تمنحنا حرية مطلقة. من الناحية النظرية، باستخدام لغة مثلC++ ، يمكننا كتابة برنامج يفعل أي شيء يمكن للحاسوب أن يفعله.

- **الميزة :**يمكننا أن نصف "أي شيء". يمكننا استخدامها لكتابة خوارزميات معقدة للرؤية الحاسوبية، وأنظمة تحكم دقيقة للمحركات، وواجهات مستخدم رسومية، كل ذلك بنفس اللغة. إنها تمنحنا قوة هائلة.
- **المشكلة الكبرى :**هذه الحرية تأتي بثمن باهظ. بسبب تعقيد اللغة الهائل (المؤشرات pointers، وإدارة الذاكرة، وmultiple inheritance) ، يصبح من **المستحيل تقريباً** على الكمبيوتر أن يحلل برنامج C++ كبير **ويثبت رياضياً** أنه خالٍ من أنواع معينة من الأخطاء. التحقق الرسمي منه شبه مستحيل.

مثال توضيحي :لنتخيل أننا أعطينا رساماً فرشاة سحرية يمكنها رسم أي شيء يخطر بباله (قوة تعبيرية هائلة)، لكن الحبر الذي تستخدمه هذه الفرشاة لا يمكن قراءته بواسطة أي ماسح ضوئي أو كمبيوتر. الرسام لديه حرية كاملة، لكن لا يمكننا أبداً التحقق من رسوماته تلقائياً أو استخدامها في آلة. لغة C++ تشبه هذه الفرشاة.

---

2. لغات ذات قوة تعبيرية محدودة (مبنية على نموذج حوسبة محدد)

هذه هي لغات النمذجة المتخصصة التي تحدثنا عنها (مثل مخططات الحالة Statecharts أو مخططات تدفق البيانات Dataflow). هذه اللغات **مقيدة عمداً**.

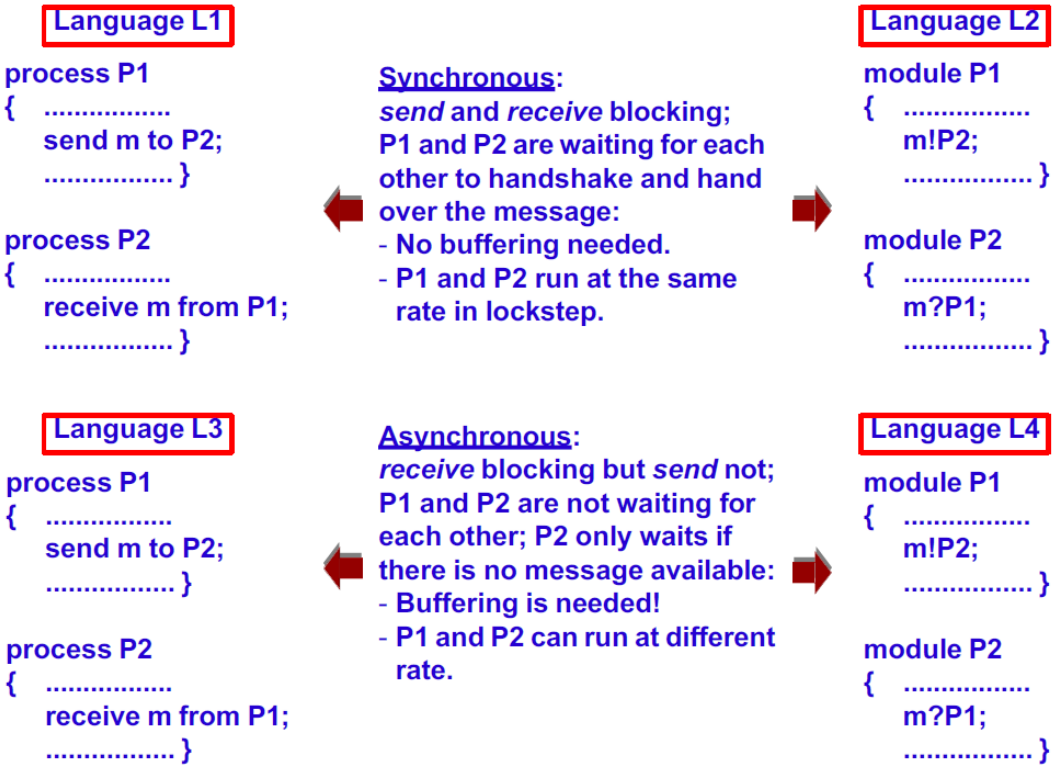
- **العيب :**لا يمكننا أن نصف بها إلا أنواعاً معينة من الأنظمة. لا يمكننا كتابة نظام رؤية حاسوبية كامل باستخدام مخطط حالة فقط. قوتها التعبيرية محدودة.
- **المزايا الجبارة :**هذا القيد هو مصدر قوتها لأن اللغة بسيطة ولها قواعد ومعنى (semantics) واضح جداً، فإنها تمنحنا فائدتين هائلتين:
  1. **التحقق الرسمي ممكن :** يمكن لأدوات الكمبيوتر تحليل النموذج بسهولة و**إثبات** خصائص مهمة. يمكننا إثبات أن الروبوت لن يدخل أبداً في حالة "تجمد (deadlock) "أو أن إجراءات الأمان ستعمل دائماً.
  2. **التوليف الفعال ممكن :** يمكن للكمبيوتر أن يأخذ هذا النموذج المحدود ويقوم **بتوليد كود أو تصميم هاردوير فعال تلقائياً**.

مثال توضيحي :لنتخيل أننا أعطينا مهندساً مجموعة من مكعبات الليغو (LEGO) المصممة لبناء الجسور فقط (قوة تعبيرية محدودة). لا يمكنه بناء سيارة أو طائرة بها .**لكن**، لأن كل القطع قياسية، يمكن للكمبيوتر بسهولة أن:

1. يحلل التصميم ويتأكد من أن الجسر متين ولن ينهار(**التحقق الرسمي**)
2. يقرأ المخطط ويرسل التعليمات لذراع آلية لتقوم ببناء الجسر تلقائياً (**التوليف**)

الخلاصة :المقايضة واضحة :**الحرية المطلقة مقابل الأمان والأتمتة**.

في الأنظمة الحرجة التي لا مجال فيها للخطأ (مثل الروبوتات الطبية، أو أنظمة الفرامل في السيارات، أو الروبوتات في المصانع)، غالباً ما نفضل **التضحية ببعض الحرية** واستخدام لغات محدودة وقابلة للتحليل لضمان أن النظام سيعمل بشكل صحيح وآمن 100.%



هنا سنأخذ المفاهيم النظرية التي ناقشناها حول "المعنى (Semantics) "و"نماذج الحوسبة" ونطبقها على مشكلة حقيقية جداً في عالم الروبوتات :كيف تتواصل المهام المختلفة مع بعضها البعض؟

نعرض نموذجين مختلفين للحوسبة (طريقتين للتفكير) في عملية التواصل :المتزامن (Synchronous) وغير المتزامن (Asynchronous). نلاحظ كيف أن الكود في اليسار(L1 و L3) قد يبدو متشابهاً، لكن "معناه" مختلف تماماً.

---

1. التواصل المتزامن (Synchronous) .: طريقة المصافحة

لكي تتم عملية نقل الرسالة، يجب أن يكون كل من المُرسِل (Process P1) والمُستقبِل (Process P2) جاهزين وموجودين في نفس اللحظة بالضبط.

- السلوك:
  - عندما يصل P1 إلى أمر send m to P2، فإنه يتوقف وينتظر (blocking) . لن يكمل عمله حتى يكون P2 مستعداً لاستلام الرسالة.
  - عندما يصل P2 إلى أمر receive m from P1، فإنه أيضاً يتوقف وينتظر (blocking) حتى يقوم P1 بإرسال الرسالة.
  - عندما يكون كلاهما جاهزاً، تحدث "المصافحة"، وتنتقل الرسالة m بشكل فوري، ثم يكمل كل منهما عمله.
- مثال :لنتخيل خط تجميع فيه ذراع روبوتية (P1) تضع قطعة على سير متحرك، ومحطة لحام (P2) تقوم بلحامها .في النموذج المتزامن، عندما تضع الذراع (P1) القطعة، سترسل رسالة "القطعة جاهزة" ثم

ستتجمد في مكانها .لن تذهب لإحضار قطعة جديدة. في نفس الوقت، محطة اللحام (P2) تكون منتظرة لهذه الرسالة. عندما تصل الرسالة، تقوم P2 باللحام. فقط بعد أن تتأكد P1 أن P2 ستلمت الرسالة، تتحرك P1 لإحضار القطعة التالية.

- النتائج:
  - لا حاجة لذاكرة مؤقتة (No buffering) : الرسالة تنتقل مباشرة من يد ليد.
  - يعملان بنفس السرعة (Lockstep) : سرعة الذراع (P1) أصبحت مرتبطة تماماً بسرعة محطة اللحام (P2) . النظام كله يسير بسرعة أبطأ مكون فيه.

---

2. التواصل غير المتزامن (Asynchronous) : طريقة صندوق البريد

يمكن أن نفكر في هذا النوع على أنه إرسال بريد إلكتروني أو رسالة نصية. نحن كمُرسِل (P1) نضع الرسالة في "صندوق بريد" ونكمل عملنا فوراً. لا ننتظر لنرى هل قرأها الطرف الآخر أم لا.

- السلوك:
  - عندما يصل P1 إلى أمر send m to P2، فإنه يضع الرسالة mفي طابور (queue) أو ذاكرة مؤقتة (buffer)ثم يكمل عمله فوراً(non-blocking)
  - عندما يصل P2 إلى أمر receive m from P1، فإنه يذهب لـ"صندوق البريد". إذا وجد رسالة، يأخذها ويكمل عمله. إذا كان الصندوق فارغاً، فإنه ينتظر (blocking) حتى تصل رسالة.
- مثال :في نفس سيناريو خط التجميع. في النموذج غير المتزامن، عندما تضع الذراع (P1) القطعة، سترسل رسالة "القطعة جاهزة" ثم ستذهب فوراً لإحضار القطعة التالية دون انتظار. الرسالة يتم تخزينها في ذاكرة. محطة اللحام(P2) ، عندما تنهي عملها، تذهب وتفحص "صندوق الرسائل". إذا وجدت رسالة "القطعة جاهزة"، تقوم بعملية اللحام.
- النتائج:
  - نحتاج لذاكرة مؤقتة(Buffering is needed!) : هذا هو الشرط الأساسي. يجب أن يكون هناك مكان لتخزين الرسائل.
  - يمكن أن يعملا بسرعات مختلفة :الذراع السريعة (P1) يمكن أن تضع عدة قطع وتملأ "صندوق البريد" بالرسائل، بينما تعمل محطة اللحام البطيئة (P2) على تفريغ هذا الصندوق حسب سرعتها. هذا يزيد من كفاءة النظام بشكل عام.

ملاحظة على لغات L2 وL4 :

هذه مجرد طريقة كتابة مختصرة ورسمية. الرمز !يعني إرسال، والرمز ? يعني استقبال . نلاحظ أن نفس الرموزلها معنى (semantics) مختلف تماماً في النموذج العلوي والسفلي.

الخلاصة :الاختيار بين النموذجين هو قرار تصميمي حاسم. النموذج المتزامن أبسط و أكثر قابلية للتنبؤ، لكنه قد يكون غير فعال. النموذج غير المتزامن أكثر كفاءة ومرونة، لكنه يتطلب إدارة للذاكرة وقد يكون أعقد في تصميمه (ماذا نفعل لو امتلأ صندوق البريد؟).