Evaluation Criteria for Holonic Control Implementations in Manufacturing Systems

Karel Kruger, Anton H. Basson

International Journal of Computer Integrated Manufacturing(2018)

vol. 32, Issue 2, pp.148-158, DOI: 10.1080/0951192X.2018.1550674

Smart Factory 논문리뷰
신승준 교수님
한양대학교 산업 데이터 엔지니어링학과
석사과정 강병모

한양대학교

Introduction

- 현대 제조업은 리드 타임(Lead Time) 단축이 KEY POINT!
  - ✓ 적은 리드 타임→ 제품 변형을 통해 다품종 대량생산
- RMS(Reconfigurable Manufacturing systems), CPPS(Cyber-Physical Production Systems)
  - ✓ RMS: 특정 제품 군 내에서 (하드웨어/소프트웨어) 기능 요소를 추가하거나 제거하여 생산을 빠르게 하는 시스템
  - ✓ CPPS: 주변환경으로 부터 정보를 습득하고 자율적으로 행동 (제조 자율화)
    - →시스템의 다른 요소와 인터넷에서 사용할 수 있는 지식과 서비스에 접근 가능
    - ➔내부 및 외부 변화에 대한 대응력 갖춤
      - ➔HMS가 CPPS와 가장 부합

Figure 1: Lead Time
출처:https://www.creativesafetysupply.com/glossary/lead-time/

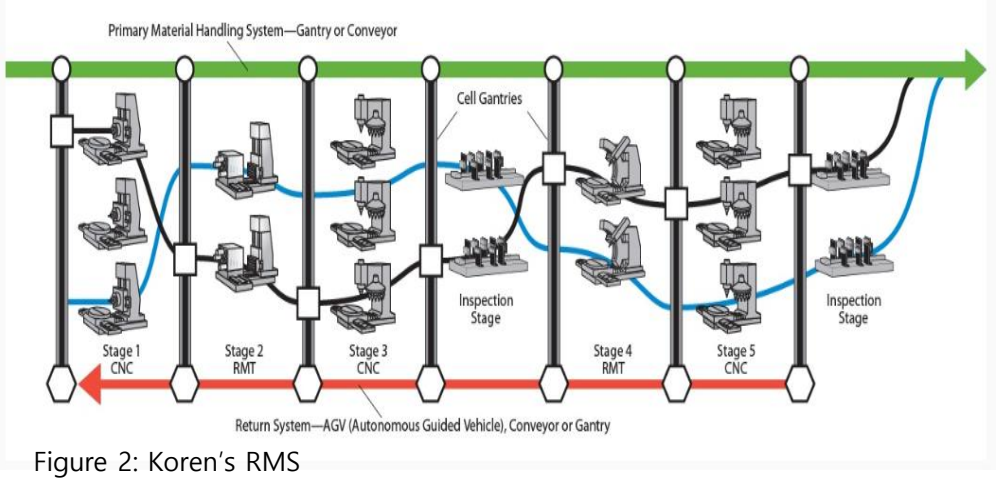

Figure 2: Koren's RMS
출처:https://www.creativesafetysupply.com/glossary/lead-time/

Introduction

- **HMS(Holonic Manufacturing System)**
  - ✓ **RMS와 CPPS를 구현 및 제어하기 위해 사용**
  - ✓ **각 부분에 holon이 존재하면서 자율적이고 독립적이지만 전체의 동작에 연결되어 있음**
    - **➔HMS : 각각 독립되어 있는 holon들의 통합을 통해 목적 지향적 유연 시스템**
  - ✓ **Architectures -PROSA : 3가지 유형의 holon의 구성을 통해 제조 시스템의 자율적 상호 운용 가능한 구조**
    - **-ACADA : Agile & Adaptive 제조를 위한 구조**
      - **➔중앙 집중식 구조와 분산 구조를 동적으로 균형을 맞춰 생산 최적화 및 민첩한 반응이 가능한 구조(수요변화에 대응)**

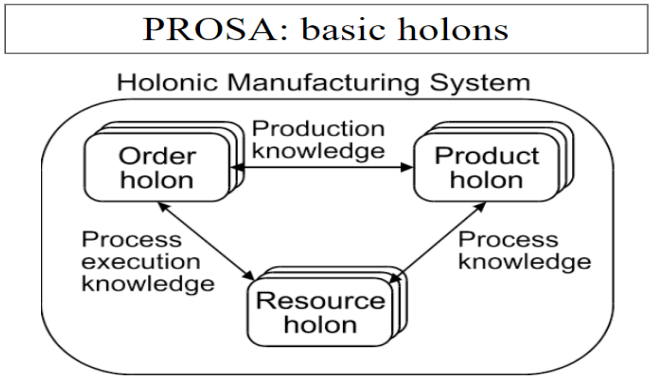

Figure 3 : PROSA(Reference architecture for holonic manufacturing systems) HMS

출처:https://www.creativesafetysupply.com/glossary/lead-time/

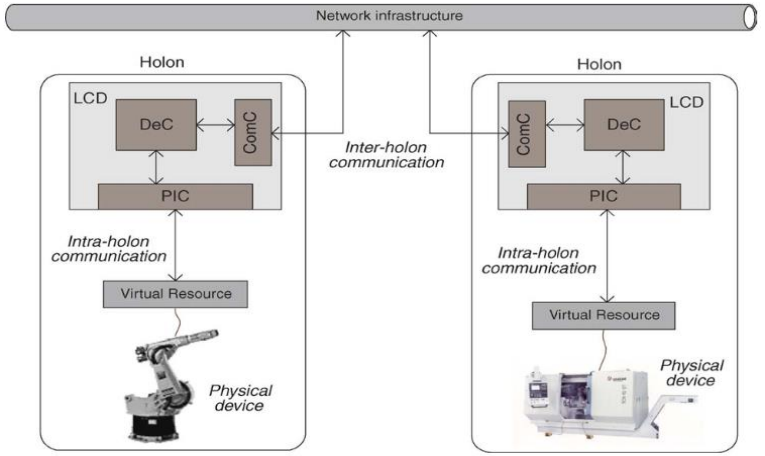

Figure 4: ACADA HMS

출처:https://www.semanticscholar.org/paper/ADACOR%3A-A-holonic-architecture-for-agile-and-Leit%C3%A3o-Restivo/7e5ee81ac7007b1d136fed38c36e70cdae739a0f/figure/0

Introduction

- 전통적인 HMS 구축 시스템
  - Agent Base Programming Platform
    - ✓ Agent : 자율적 상호 운용 가능한 객체 ≒ holon
    - ✓ Multi-agent systems(MASs) : 여러 개의 에이전트들로 구성되어 있는 시스템
      ex) PROSA, ACADA
      →가장 보편적이고 널리 사용됨
    - ✓ 이점 : 1. 분산된 자율성– 에이전트들이 독립적으로 행동→제조 공정의 유연성과 효율성 UP!
      2. 안정성- 분산된 구조로 인해, 각 에이전트가 오류를 감지하고 대처
      3. 적응성- 환경이 변화해도 자율적으로 대처 가능→유연하고 향상된 생산성을 발휘
      4. 확장성- 에이전트들이 독립적으로 작동함으로, 새로운 에이전트를 추가 용이

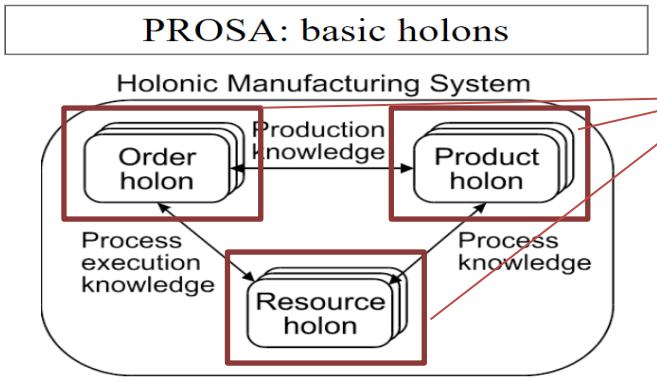

Figure 3 : PROSA(Reference architecture for holonic manufacturing systems) HMS
출처:https://www.creativesafetysupply.com/glossary/lead-time/

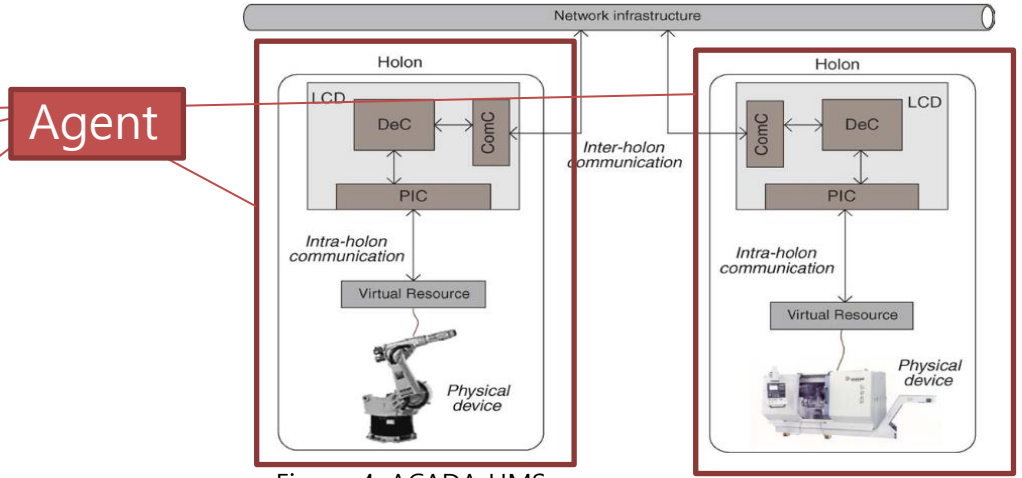

Figure 4: ACADA HMS
출처:https://www.semanticscholar.org/paper/ADACOR%3A-A-holonic-architecture-for-agile-and-Leit%C3%A3o-Restivo/7e5ee81ac7007b1d136fed38c36e70cdae739a0f/figure/0

Introduction

- **Agent Base Programming Platform**
  - ✓ 단점 -새로운 기술을 도입하기 위해서는 코드/시스템을 재구성 해야함
    →에이전트 기반 프로그램의 코드가 길고 복잡함
    →코드를 재작성하기 위해서는 시간과 비용이 많이 발생함
    ➔산업계의 요구 : 적은 리드 타임을 통한 다품종 대량생산
    →Erlang/OTP(Open Telecom Platform)와 같은 대체 언어/플랫폼들이 대안으로 떠오름
    OTP: 각각의 프로세스로 구성되며, 프로세스들 간의 통신을 통해 프로세스들은 런타임 동안 동작이 변경 가능한 플랫폼

Figure 5: Erlang
출처:https://www.erlang.org/

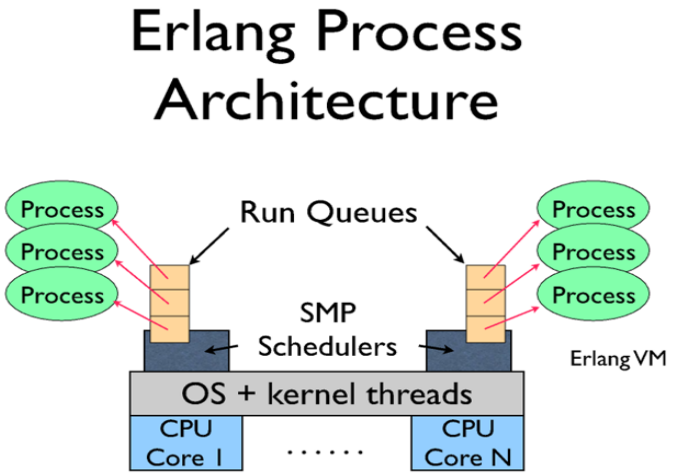

Figure 6: Erlang/OTP
출처:https://www.infoq.com/presentations/Erlang-OTP-Behaviors/

Purpose of Paper

- 산업체에 맞는 HMS 구현 플랫폼과 언어를 선택할 때 가이드라인을 제시
  - ✓ 이전 연구들에서 HMS 구현 플랫폼과 언어 평가에 대한 기준을 제시하였지만, 평가 기준의 초점과 관점이 다양해서 어떠한 플랫폼/언어를 선택하는 것에 어려움

- 산업계의 요구에 부합하는 특성을 정량적 및 정성적으로 측정하여 제시
  - ✓ 기존 평가 및 연구들은 암묵적 지식에 그쳤지만, 저자의 산업 제조 경험 및 문헌 연구를 통해 형식적 지식으로 구현함

Contributions

➔ HMS을 고려하는 산업체들에게 아키텍처, 플랫폼 선택시 가이드라인 제공
→산업체에 맞는 아키텍처, 플랫폼을 선택할 수 있게 함

➔ 산업체가 맞춤형 HMS을 개발하여 산업계에 활용될 수 있음

Industrial Needs for Hardware

- 산업계가 HMS를 채택 시 원하는 두가지 요구: 가용성, 지원가능성
  - ✓ 가용성(Availability): 생산 장비 및 시스템이 사용 가능하고 완전히 작동하는 시간의 비율
    →최적이 아닌 상태에서도 생산할 수 있는 시간의 비율
  - ✓ 지원가능성(Supportability) : HMS 도입시 유지관리가 용이한가
    →오류 발생 시간을 줄일 수 있나

➔ 가용성, 지원가능성을 극대화 시켜 생산 시간을 늘릴 수 있냐!

→가용성, 지원 가능성을 높이려면 재구성 가능성(Reconfigurability), 견고성(robustness) 유지보수성(maintainability)를 충족시켜야함

- 재구성 가능성 : 제조 시스템에서 기능 개체를 추가/제거하여 재배치 작업을 할 수 있는 능력
  ➔다품종 대량 생산에 필수적인 요소
- 견고성 : 예기치 않은 장애/오류 발생시 생산에 사용할 수 있는 상태를 유지할 수 있는 능력
  →예기치 못한 장애 발생에도 생산 가능을 유지함
- 유지보수성 : 소프트웨어를 유지보수를 할 수 있는 능력
  →결함을 수정/성능 개선
  ➔유지보수성이 우수→생산 중단 시간과 비용 감소
  ➔가용성과 지원가능상 향상

Industrial Needs for Software

- **HMS 개발자가 원하는 한가지 요구- 개발 생산성**
  - ✓ **개발 생산성(Development Productivity)-HMS 소프트웨어의 코드를 쉽게 개발하고 생성 할 수 있는 능력**

→개발 생산성을 높이기 위해서는 소프트웨어 복잡성(software complexity),소프트웨어 재사용(reuse of software)을 충족시켜야함

- **소프트웨어 복잡성: 얼마나 구현할 수 있는지, 얼마나 시스템을 이해하는지, 수정 및 유지 관리의 어려움을 나타낼 수 있는지**
- **소프트웨어 재사용: 새로운 소프트웨어 시스템을 구축할 때, 기존 소프트웨어 아티팩트(artefact)를 얼마나 사용할 수 있는가**

→코드를 재사용할 수 있으면 코드를 줄여 개발 생산성을 높임

➔소프트웨어 복잡성, 재사용, 검증을 통해 개발자가 코드를 얼마나 효율적이고 용이하게 사용함으로서 개발 생산성을 높일 수 있는가

***아티팩트(artefact): 소프트웨어 개발 과정에서 생산된 모든 산출물**

Evaluation metrics

- 산업계의 요구사항(재구성가능성, 견고성, 유지보수성, 컨트롤러 요구사항)과 개발자의 요구사항(소프트웨어 복잡성, 재사용 가능성) 및 소프트웨어의 검증을 정량적, 정성적으로 측정함

Table 1. Relationships between requirements and performance measures.

			Characteristics						
			Availability Supportability Development Productivity						
			Requirements						
			Reconfigurability	Robustness	Maintainability	Controller requirements	Complexity	Verification	Reusability
Performance measures	Quantitative	Reconfiguration time	*				*	*	*
		Development time					*	*	*
		Code complexity			*		*		
		Code extension rate	*		*		*		
		Code reuse rate	*		*		*		*
		Computational resource requirements				*			
	Qualitative	Modularity	*		*			*	*
		Integrability	*						*
		Diagnosability	*	*	*			*	
		Convertibility	*		*				
		Fault tolerance		*					
		Distributability				*			
		Developer training requirements			*		*	*	

Performance Measure-Quantitative

- **Quantitative Measure-산업계와 개발자들이 원하는 요구사항을 정량적으로 측정**
  (재구성 시간, 개발시간, 코드 복잡성, 코드 확장성, 코드 재사용률, 컴퓨팅 리소스 요구사항)
  ➔소프트웨어 측정 측면이 강함

  - ✓ **재구성 시간(reconfiguration time) for (재구성 가능성, 코드 복잡성, 재사용성, 검증)**
    - 재구성 시간 : 재구성을 수행하는 데 걸리는 시간
    ➔개발자가 새로운 홀론을 효과적으로 활용하기 위해 시스템을 조정 하는데 필요한 시간
    (변경 사항을 구현하는 데 필요한 시간+시스템이 필요에 따라 수행되는지 검증에 필요한 시간)

  - ✓ **개발 시간(development time) for (코드 복잡성,재사용성, 검증)**
    -개발 시간 : 새로운 제어 소프트웨어를 개발하는데 필요한 시간
    ➔새로운 유형의 홀론을 개발하여 기존 홀론 시스템에 추가에 걸리는 시간
    →개발자의 근무 시간(순수 코드를 작업하는 시간)

Performance Measure-Quantitative

✓ 코드 복잡성(code complexity) for (유지보수성, 코드 복잡성)

- 개발 생산성에 미치는 영향 중 하나→코드 복잡성
- 코드 복잡할 수록 산업계 채택의 장벽이 커짐

(산업계가 multi-agent base시스템의 단점이라고 생각했던 것!)

-이전 방식 : 사이클로매틱수(cyclomatic number)

- 코드에서 분기, 루프, 조건문 등의 제어 구조가 얼마나 많이 사용했는가를 측정

→알고리즘의 복잡성(과정의 복잡성)

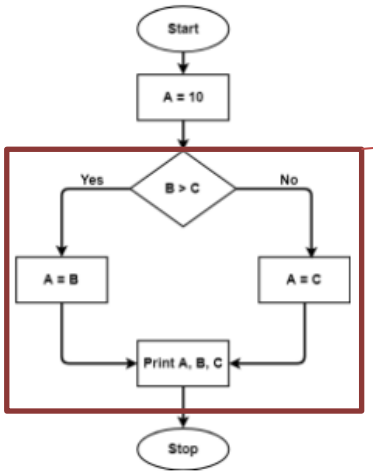

Figure 6: 사이클로매틱수

출처:https://www.geeksforgeeks.org/cyclomatic-complexity/

- 사이클로매틱수 계산 방법
  복잡도(M)=연결 선 수(E=7)-노드 수(N=7)+2*반복 cycle의 수(P=1)
  = 2
- 논문 제안 방법 : 결과 소스 코드의 복잡성 측정(SLOC)
  →HMS에서 사이클로매틱수로 복잡성을 측정하게 되면,
  서로 다른 파트의 의사결정을 다루는 총체적 과정을 다뤄야함
  ➔ 너무 복잡해짐→객관화 하기 힘듦
  - SLOC(Source Line of Code) : 결과 소스의 길이로 판단하는 지표
    →일반적으로 결과 소스의 길이가 길어질수록 작업 시간이 늘어나고
    오류가 많이 발생함

Performance Measure-Quantitative

✓ 코드 확장률(code extension rate) for (재구성 가능성, 유지보수성, 코드복잡성)
- HMS를 요구에 맞게 재구성하기 위해서는 소스 코드도 재구성 하는 것이 필요함
→새로운 기능 요구 사항을 충족하기 위해 기존 구현된 소스를 이용하여 얼마나 쉽게 조정할 수 있는가.

$$E_{i+1} = \frac{SLOC_{i+1}}{SLOC_i} \rightarrow \mathrm{i+1의\ 코드\ 확장률} = \frac{i+1의\ 코드길이}{i의\ 코드길이}$$

➔코드 확장률 : 이전의 코드보다 얼마나 증가되었는가?

✓ 코드 재사용률(development time) for (재구성 가능성, 유지보수성,코드 복잡성, 재사용성)
-개발자의 생산성 증가를 위해 기존에 존재하는 코드를 얼마나 사용할 수 있는지 중요

$$R_{i+1} = \frac{SLOC_i}{SLOC_{i+1}} \rightarrow \mathrm{i+1의\ 코드\ 재사용률} = \frac{i의\ 코드길이}{i+1의\ 코드길이}$$

➔코드 재사용률 : 이전의 코드를 이용하였는가?

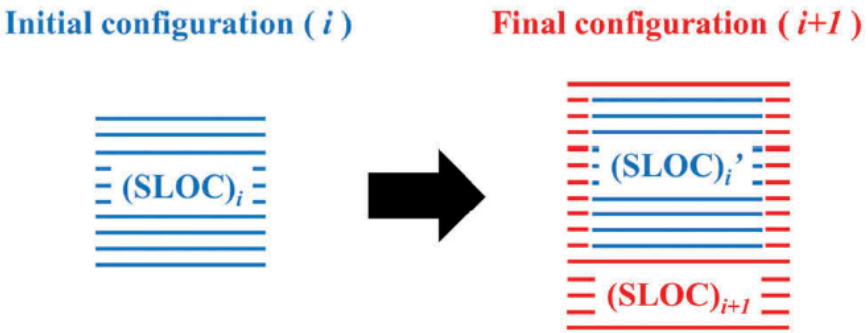

Figure 7: 코드 확장률 & 재사용률
출처: Karel Kruger, Anton H. Basson (2018)

Performance Measure-Quantitative

✓ 컴퓨팅 리소스 요구사항(computational resource requirements))

-컨트롤러에 필요한 컴퓨팅 리소스(계산 능력 및 메모리 용량)

-컨트롤러의 역할 : 소프트웨어를 통해 하드웨어를 실행시키는 것!

-key point : 계산 능력 및 메모리 용량

- CPU 시간: CPU 사용량

  →소프트웨어 명령 시간 +하드웨어 실행 시간

  ➔생산 시작 시간~생산 완료 시간

- 메모리 사용량

  -생산되는 동안 실행되는 RAM을 계산

  ➔ 생산 시작~생산 완료까지 사용되는 메모리 사용량

Performance Measure-Qualitative

- **Qualitative Measure-산업계와 개발자들이 원하는 요구사항을 정성적으로 측정**

  (모듈화, 통합성, 진단 가능성, 전환 가능성, 내결함성, 분배성, 개발자 트레이닝 요구사항)

  ➔하드웨어/실행 측면이 강함

  - ✓ **모듈화(Modularity) for (재구성 가능성, 유지보수성, 코드 재사용 가능성, 검증)**

    -모듈화 : 하나의 소프트웨어를 통해 여러 개의 소프트웨어로 분할

    ➔여러가지의 프로세스를 여러 개의 소프트웨어로 처리(분업화)

    ➔시스템 전체에 영향을 주지 않으면서 시스템을 변경, 개선할 수 있음

    ➔가변적,분산적 제조의 핵심

  - ✓ **통합성(Integrability) for (재구성가능성, 재사용가능성)**

    -통합성: 기계, 정보 및 제어 구성요소를 기존 시스템과 빠르고 효과적으로 통합할 수 있는 능력

    ➔이전에 사용하던 시스템과 호환성을 유지할 수 있는가(레거시 시스템)

Performance Measure-Qualitative

✓ 진단가능성(Diagnosability) for (재구성 가능성, 견고성, 유지보수성, 검증)

-진단가능성 : 시스템에서 품질 및 신뢰성 문제의 원인을 얼마나 쉽고 빠르게 파악할 수 있는가

→ 진단가능성이 우수하면 램프업 타임과 다운타임이 줄어듦

→ 오류의 원인과 위치를 파악하고, 모니터링을 통해 오류가 발생한 곳에 기본 실행 및 매커니즘 커뮤니케이션 실행

➔ CPPS와 industry 4.0의 주요 특성

✓ 전환 가능성(Convertibility) for (재구성가능성, 유지보수성)

-전환 가능성: 새로운 생산 요구 사항을 충족하기 위해 기존 시스템의 기능을 변환할 수 있는 능력

ex)생산 일정 변경(긴급주문), 예기치 못한 유지보수 등

→ 전환가능성이 우수하면 재구성 시간과 시스템 수명 주기 비용이 줄어듦

➔ 시스템의 생산성을 높일 수 있음

※램프업 타임: 새로운 제품을 처음 생산하기 위해 설비 공정등을 적용하기 시작하여 정상 생산에 도달하는 시간
다운타임: 제조 공정에서 설비, 기계, 시스템등이 정상적으로 작동하지 않아 생산이 중단되는 시간

Performance Measure-Qualitative

- ✓ 내결함성(Fault tolerance) for (견고성)
  -제조 시스템에서 오류가 발생하는 것은 불가피함
  - 내결함성: 유용한 수준의 시스템 안정성을 유지하면서 작동할 수 있는 능력
  ➔오류 발생한 홀론의 오류를 차단해서 다른 홀론에 영향을 주지 않는 능력

- ✓ 분배성, 분산성, 배포가능성(Distributability) for (컨트롤러 요구사항)
  - HMS의 특징 : 분산 제어 가능
  - 분배성 : 각각의 홀론이 자율적으로 작업을 수행하고 다른 홀론들과 협력하여
  전체 시스템을 제어하는 분산 제어 능력
  ➔분배성이 높으면, 자율적이고 협력적인 요소들을 조합하여 높은 유연성과 반응성을 갖는 제조 시스템을 구축할 수 있음

- ✓ 개발자 교육 요구 사항(Developer training requirements) for (유지보수성, 코드복잡성, 검증)
  -HMS 상대적으로 규모가 적어 개발자가 부족함
  -HMS 원리 및 구현에 대한 교육을 통해 개발자 교육→시간과 비용이 많이 듦
  ➔개발자 교육 요구 사항을 통해 유지보수성, HMS 소프트웨어 개발을 용이하게 함

Implementation Case Study

- **Case Study : 전기 회로 차단기 조립 및 품질 검사 셀의 홀로닉 제어**
  1. **Manual Assembly: 회로 차단기의 부품을 사람이 조립**
  2. **Inspection : 조립된 회로 차단기를 검사**
     →검사방법 : 머신비전 검사 –컴퓨터 비전 기술을 이용하여
     부품의 오류가 있는지 없는지 검사
     →정확도가 높고, 고속으로 처리 가능함
  3. **Electrical test(ETS) : 조립된 전기회로 차단기가 잘 작동하는지 검사**
  4. **Stacking – 여러 차단기를 업체의 요구에 맞게 단극/양극 회로 차단기를 생산함**
     ➔지속적으로 나온 재구성!
  5. **Riveting – 두개 이상의 재료를 고정시키기 위해 사용되는 고정 방법**
     ➔Stacking을 통해 쌓았으면, riveting을 통해 고강도 재료를 연결
  6. **Removal – 완성된 회로 차단기가 캐리어에서 제거→포장을 위해 다음으로 넘어감**

→PROSA 구조를 참조하여 Erlang/OTP를 사용하여 HMS 구현

※ Assurance cell: 제조공정에서 발생하는 데이터와 정보를 수집하고 분석하는 역할 수행

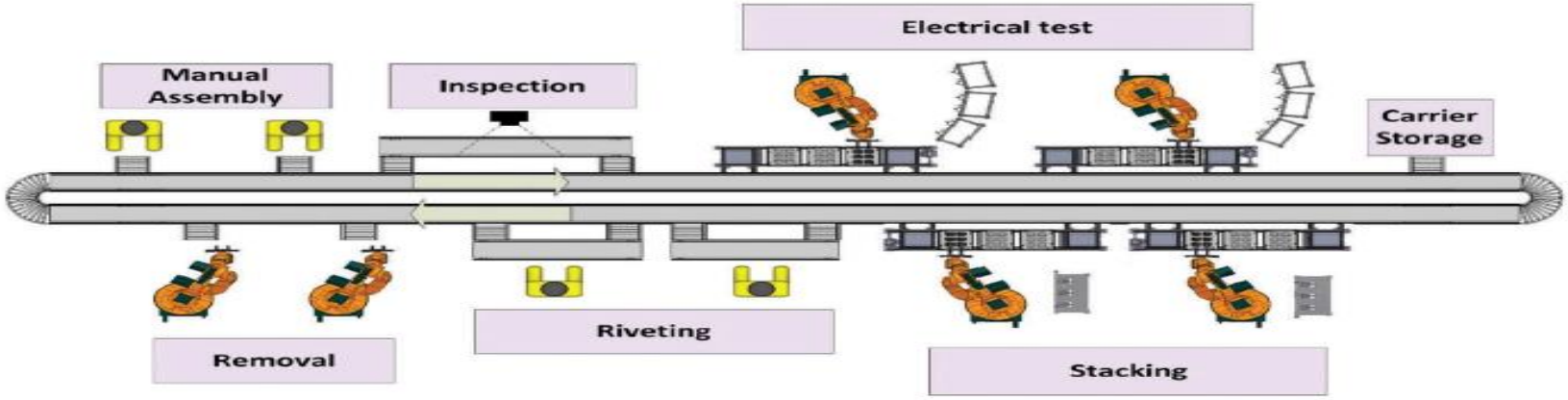

Figure 8: Layout of electrical circuit breaker assembly and quality assurance cell
출처: Karel Kruger, Anton H. Basson (2018)

Implementation Case Study Results

- **Quantitative measures**
  - ✓ 재구성 실험 : Stacking 리소스 홀론을 통합하기 위해 제품 및 주문 홀론에 대한 변경
    ➔고객이 단극을 주문했는지, 양극을 주문했는지에 따라, stacking 부분 홀론에 적용하여 재구성에 맞는 환경이 조성되었는지 시간으로 측정(h)
  - ✓ 개발 시간 : Stacking 홀론에서 단극에서 양극(또는 그 반대)로 전환할때 효과적으로 전환하기 위한 코드를 작성하고 개발하는데 걸린 시간으로 측정(h)
  - ✓ 코드 복잡성 : 주문 홀론과 ETS에 필요한 코드에 대한 복잡성 측정(SLOC)
  - ✓ 코드 확장률 & 재사용 가능성 : Stacking 홀론에서 단극 주문 코드에서 양극 주문 코드가 얼마나 확장되고 재사용되었는지 측정(SLOC)
  - ✓ 컴퓨팅 리소스 요구사항 : 생산/주문 홀론에 사용된 CPU 코어/메모리 사용량/CPU 시간

➔ 주관적 요소가 적기 때문에 엄청난 가치를 얻을 수 있음
However, 아키텍처/플랫폼의 결과 비교가 있어야만 유의미한 결과를 얻을 수 있음
→ 미래의 연구에서는 정량적 결과의 기준점을 도출하는 것이 필요할 듯

Figure 9: 단극/양극 회로 차단기

Table 2. Quantitative measurements obtained from the case study control implementation.

Quantitative measure		Value	Unit
Reconfiguration time		1.8	h
Development time		2.5	h
Code complexity	Order holon	318	SLOC
	ETS Resource holon	189	SLOC
Code extension rate		1.2	
Code reuse rate		0.77	
Computational resource requirements	Thread count	28	
	RAM usage	24.1	MB
	CPU time	7.8	%

Implementation Case Study Results

➢ Qualitative measures

Table 3. Summary of the qualitative evaluation of the case study.

Qualitative measure		Evaluation
Modularity	Architecture	Modules are containers of program code, and all code is structured as functions. Includes OTP, which is a set of robust libraries that can be used to structure applications.
	Module interaction	Functions can be exported from modules, so that it can be called from other modules.
	Testing	All exported functions can be individually tested. A shell process can be started, which provides the developer with an interface for giving inputs to executing processes. The observer application can set up to trace process communication and execution.
Integrability	Integration of foreign code	Foreign code can be represented as a process-like object, called a port. Ports allow different integration mechanisms with foreign code.
	Support for communication protocols	Libraries are available for commonly used protocols, such as serial, TCP and UDP. XMErL library also available for building and parsing XML documents.
Diagnosability	Shell processes and observer application can be used to gather information and test the interaction between processes. The observer application can also monitor process execution by tracing function calls and messaging.	
Convertibility	The developer can use shell processes to start and stop the execution of other processes. Changes can be updated to running code without the need to stop execution (i.e. hot code loading).	
Fault tolerance	The Erlang process model allows for the isolation of errors – processes, as the basic unit of abstraction, act as boundaries that limit the propagation of errors. Also, Erlang/OTP facilitates the creation of supervision trees, where supervisor processes monitor worker processes and trap occurring errors.	
Distributability	Distributable architecture	Nodes (instances of the Erlang virtual machine that are configured for networking) constitute the architectural provision for distribution.
	Communication in distribution	Distributed nodes are discovered by the Erlang Port Mapper Daemon, which allows for communication between processes on different nodes. The messaging mechanism functions unchanged in distributed configurations.
	Tools for distribution	Debugging and monitoring tools are equally useful in the distributed configuration.
	Portability	Erlang virtual machine allows for platform independence, and is supported on the most prominent operating systems.
	Standardization and guidelines	OTP behaviours offer a standardized structure for code. No standards for communication – the developer has complete freedom.
Developer training requirements	Holon behaviour	Using OTP libraries, and distributing holon functionality over several lightweight processes, provides structure for the implementation of holon behaviour.
	Inter-holon communication	Native messaging mechanism and functions facilitates the communication between processes.
	External communication	Libraries for communication protocols are available.
	Concurrency	Process model allows for many concurrent, lightweight processes to be created and managed.
	Verification of functionality	Shell processes and tracing tools allow for verification with a high level of granularity.

Implementation Case Study Results

- **Qualitative measures**
  - ✓ 모듈성 - 구조 – 모듈을 함수로 짜여진 코드로 구성할 수 있느냐 없느냐(o/x)
    - 모듈과의 상호작용 – 모듈과 모듈 사이에 상호교환이 되느냐(o/x)
    - 테스트 – 모듈마다 구성되어 있는 코드이 실행가능 하고 다른 모듈들과 정상적으로 상호작용이 되는가(o/x)
  - ✓ 통합성 - 다른 코드와의 결합여부-외부코드를 소프트웨어에 통합시켜 HMS을 실행시킬 수 있는가 (o/x)
    - 통신 프로토콜 지원여부-TCP, UDP와 같은 프로토콜을 지원할 수 있는가(o/x)
  - ✓ 진단가능성 - assurance cell을 통해 정보를 수집하여 오류가 발생할 가능성을 진단 할 수 있는가 (o/x)
  - ✓ 전환가능성 - 예기치 못한 변수, 상황에 시스템이 잘 작동하는지 (o/x)
  - ✓ 내결함성 – 특정 플랫폼/언어를 사용했을 때, 오류 발생 홀론의 오류를 차단하여 주변 시스템에 오류 발생에 영향을 미치지 않는지 여부(o/x)
  - ✓ 분산성 - 분산 구조 지원가능성-특정 플랫폼/언어가 분산된 홀론들을 실행시킬 수 있는가(o/x)
    - 분산의 상호작용 – 특정 플랫폼/언어가 분산된 홀론들을 연결시켜줄 수 있는가(o/x)
    - 분산을 위한 도구 – 특정 플랫폼/언어가 분산구조에서 디버깅 및 모니터링이 되는가(o/x)
    - 휴대성 – 특정 플랫폼이 분산된 구조를 지원함을 통해 독립적인 운영시스템을 구축 할 수 있는가(o/x)
    - 표준화 및 가이드라인-특정 플랫폼이 표준화된 코드 구조를 제공할 수 있는가(o/x)
  - ✓ 개발자 교육 요구 사항 - 홀론을 이해하고 특정 플랫폼/언어를 사용하여 구현 할 수 있는가(o/x)
    - 내부 홀론들간의 관계를 이해하고 조작하고 다룰수 있는가(o/x)
    - 외부 홀론들을 연결하고 조작할 수 있는가(o/x)
    - 특정 언어를 통해 구현한 홀론 시스템을 보장하고 검증할 수 있는가(o/x)

➔정성적 지표를 통해 HMS 구현에 있어 전반적인 통찰력을 제공해 줄 수 있음

Conclusion

- 현대 제조 산업은 리드타임 단축을 통해 다품종 대량 생산 하는 것을 목표로 함
  →RMS/CPPS를 구현하기 위해 HMS 구조가 적합함

- HMS 구조를 산업계에서 사용하기 위해서는 가용성과 지원가능성의 증명이 필요
  →가용성과 지원가능성을 증명할 지표들이 필요했음
  →Agent 기반 프로그래밍 시스템 및 Erlang/OTP등 다양한 시스템 속에서 산업체에 맞는 플랫폼과 언어를 선택할 기준이 명확하지 않았음

- 정량적(6가지 요소)/정성적(7가지 요소)들을 통해 특정 플랫폼/언어를 사용할 때 가용성, 지원가능성, 개발 생산성을 지원하는지 Case Study를 통해 보여줌

- Case study의 지표 측정은 stacking 부분 수행에 의존적임

- 정량적 지표는 주관적 요소가 적지만, 아키텍처/플랫폼의 결과 비교가 있어야만 유의미한 결과를 얻을 수 있음

- 정성적 지표는 산업체가 HMS 구현에 있어 전반적인 통찰력을 제공해 줄 수 있음

Further Study

- HMS 구현의 평가지표가 유의미하게 사용하기 위해서는 아키텍처/플랫폼의 결과 비교를 진행해야 함
  - ✓ 논문에서는 PROSA 구조를 참조하여 Erlang/OTP로 구현을 함
  - ✓ 다양한 플랫폼(Eclipse-자바기반, ROS-Robot Operating System, Matlab), 다양한 언어(Java-객체지향, C++-고성능 컴퓨팅 언어, Python)를 통해 HMS를 구현하여 결과 비교가 필요함
  - ➔ 구현된 플랫폼/언어의 결과 비교를 통해 구체화된 수치 지표를 제시할 수 있음