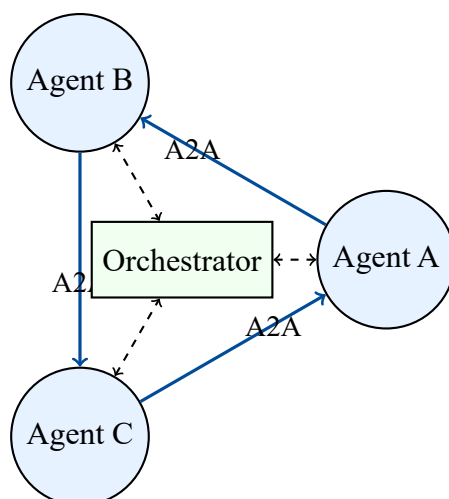


## פרק 6

### פרוטוקול A2A -- כסוכנים מדברים ביניהם

איור 1.6 מציג את הרעיון המרכזי של הפרק: סוכנים אוטונומיים המתקשרים ביניהם באמצעות פרוטוקול מובנה, תחת תיאום של מתזמר (Orchestrator) מרכזי.



איור 1.6: מערכת Multi-Agent עם מתזמר מרכזי

### יעדי הלמידה

בסיום פרק זה תוכלו:

- להבין מהו פרוטוקול A2A ואיך סוכנים מתקשרים זה עם זה
- לתכנן ארכיטקטורה של מערכת Multi-Agent
- לנהל תיאום ותזמון בין סוכנים אוטונומיים
- להתמודד עם קונפליקטים והתנגשויות בין סוכנים
- להשתמש בכלי תזמור (Orchestration) כמו LangGraph
- לנטר ולמדוד ביצועים של מערכות Multi-Agent

## 1.6 מבוא: הכוח של רבים

חברת מסחר אלקטרוני גדולה עומדת בפני אתגר: כל הזמנה שמגיעה דורשת טיפול של מספר מחלקות -- שירות לקוחות צריך לאשר את הפרטים, מחלקת המלאי צריכה לבדוק זמינות, מחלקת התמחור צריכה לאשר מחירים והנחות, ולוגיסטיקה צריכה לתכנן משלוח. בעבר, כל זה נעשה באמצעות מערכות נפרדות, אימיילים אינסופיים ושיחות טלפון. כל הזמנה לקחה שעות, לפעמים ימים.

פתרון אפשרי היה לבנות סוכן ענק אחד שמטפל בהכל. אבל הסוכן הזה היה צריך לדעת הכל -- את כל כללי המלאי, את כל מדיניות התמחור, את כל מגבלות הלוגיסטיקה. הוא היה נהיה מורכב מדי, איטי מדי, וקשה מדי לתחזוקה.

הפתרון שהחברה בחרה היה שונה לחלוטין: במקום סוכן אחד גדול, היא בנתה חמישה סוכנים קטנים, כל אחד מומחה בתחום. סוכן שירות לקוחות, סוכן מלאי, סוכן תמחור, סוכן לוגיסטיקה וסוכן תשלומים. כל אחד מהם יודע לעשות דבר אחד, אבל לעשות אותו מצוין.

האתגר האמיתי התחיל כשהחברה שאלה: איך הסוכנים האלה ידברו אחד עם השני? מי יחליט מי עושה מה? מה קורה אם שני סוכנים לא מסכימים? איך מוודאים שכולם עובדים לקראת אותה מטרה?

זהו בדיוק התפקיד של פרוטוקול Agent-to-Agent (A2A).

## 2.6 מהו A2A?

### 1.2.6 הגדרה

**הגדרה 1.6** (פרוטוקול A2A). **Agent-to-Agent Protocol (A2A)** הוא פרוטוקול תקשורת המאפשר לסוכנים אוטונומיים לתקשר זה עם זה, לחלוק מידע, לתאם פעולות ולעבוד יחד להשגת מטרה משותפת [1], [4].

בעוד ש-MCP (פרוטוקול Model Context) עוסק בתקשורת בין סוכן בודד לבין המודל שלו, A2A עוסק בתקשורת **בין סוכנים** [9], [01].

### 2.2.6 מדוע נדרש A2A?

בעולם העסקי, בעיות רבות מורכבות מדי עבור סוכן בודד:

- **מומחיות מבוזרת:** ידע שונה נמצא במקומות שונים (MRC, PRE, מלאי, לוגיסטיקה)
- **זמינות ותפוסה:** סוכן אחד לא יכול לטפל בהכל בו-זמנית
- **חוסן (Resilience):** אם סוכן אחד נכשל, אחרים יכולים להמשיך
- **התמחות:** כל סוכן יכול להתמחות במה שהוא עושה הכי טוב
- **סקלביליות:** קל יותר להוסיף סוכנים חדשים מאשר להגדיל סוכן קיים

#### דוגמה: מערכת הזמנות

במערכת הזמנות אונליין:

- **סוכן קבלה (Intake Agent):** מקבל את ההזמנה, מוודא שהנתונים תקינים
- **סוכן מלאי (Inventory Agent):** בודק זמינות מוצרים

- **סוכן תמחור (Pricing Agent):** מחשב מחיר סופי, כולל הנחות
  - **סוכן לוגיסטיקה (Logistics Agent):** מתזמן משלוח
  - **סוכן תשלום (Payment Agent):** מעבד תשלום
- כל סוכן עושה את התפקיד שלו, אבל הם חייבים לתקשר כדי להשלים את ההזמנה מקצה לקצה.

### 3.2.6 מרכיבי A2A

פרוטוקול A2A טיפוסי מורכב מהמרכיבים הבאים:

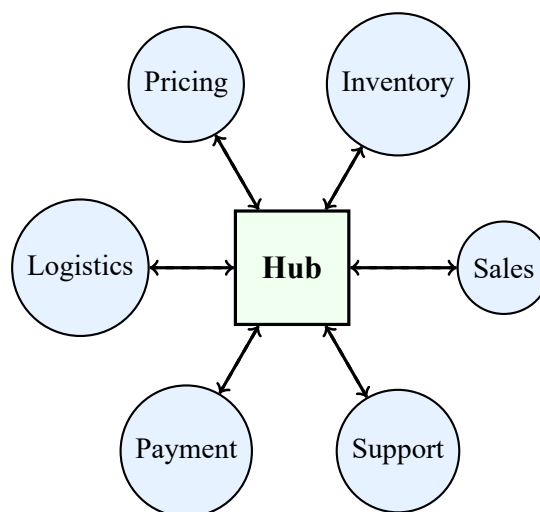
1. **שכבת תקשורת (Communication Layer):** איך הסוכנים שולחים ומקבלים הודעות
2. **פורמט הודעות (Message Format):** מבנה סטנדרטי של הודעות (לרוב JSON)
3. **ניתוב (Routing):** כיצד הודעה מגיעה לסוכן הנכון
4. **תזמור (Orchestration):** מנגנון לתיאום פעולות בין סוכנים
5. **ניהול מצב (State Management):** מעקב אחרי מצב השיחה/המשימה
6. **טיפול בשגיאות (Error Handling):** מה קורה כשסוכן נכשל

### 3.6 ארכיטקטורות של מערכות tnegA-itluM

ישנן שלוש ארכיטקטורות עיקריות למערכות Multi-Agent [3], [41]:

#### 1.3.6 ekopS-dna-buH (מרכז וזרועות)

בארכיטקטורה זו, יש סוכן מרכזי אחד (Hub) שמתאם את כל האחרים (Spokes), כפי שמוצג באיור 2.6.



איור 2.6: ארכיטקטורת Hub-and-Spoke -- סוכן מרכזי מתאם את כולם

#### יתרונות:

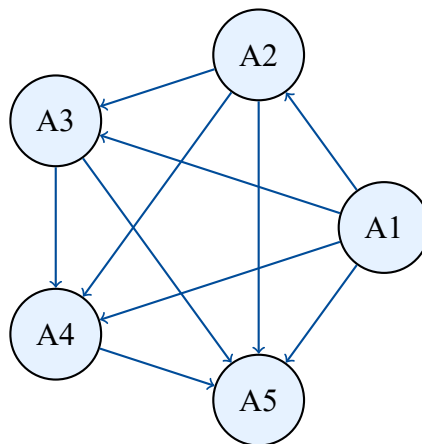
- פשוט לתכנן ולנהל
- בקרה מרכזית ברורה
- קל לדבג ולנטר

#### חסרונות:

- ה-Hub הוא נקודת כשל יחידה (Single Point of Failure)
- יכול להפוך לצוואר בקבוק (Bottleneck)
- קשה לסקלביליות גבוהה

### 2.3.6 hseM (רשת)

בארכיטקטורת רשת (Mesh), כל סוכן יכול לדבר עם כל סוכן אחר ישירות, כפי שמוצג באיור 3.6.



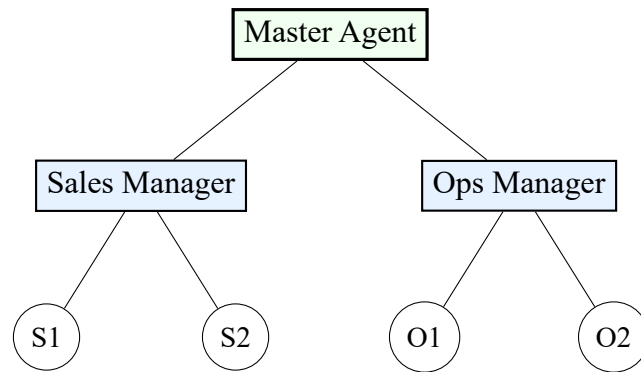
איור 3.6: ארכיטקטורת Mesh -- כל סוכן מתקשר ישירות עם כולם

#### יתרונות:

- אין נקודת כשל יחידה
- גמישות גבוהה
- תקשורת ישירה ומהירה

#### חסרונות:

- מורכב מאוד לנהל
- קשה לוודא עקביות
- קשה לעקוב אחרי זרימת מידע



איור 4.6: ארכיטקטורה היררכית -- סוכן ראשי מנהל מנהלי ביניים

### 3.3.6 lacihcrareiH (היררכי)

בארכיטקטורה היררכית, סוכנים מאורגנים בשכבות, כאשר סוכנים ברמה גבוהה יותר מנהלים סוכנים ברמה נמוכה יותר. איור 4.6 מדגים מבנה זה.

**יתרונות:**

- מבנה ברור וסדור
- קל לנהל באופן מודולרי
- מתאים לארגונים קיימים

**חסרונות:**

- יכול להיות איטי (הודעות עוברות דרך שכבות)
- נוקשה למדי
- דורש ניהול זהיר של ההיררכיה

## 4.6 תיאום משימות בין סוכנים

### 1.4.6 חלוקת עבודה (noitubirtsiD ksaT)

אחת השאלות החשובות ביותר במערכת Multi-Agent היא: איך מחלקים עבודה בין הסוכנים? ישנן מספר אסטרטגיות:

#### חלוקה סטטית (tnemngissA citatS)

כל סוכן מקבל מראש סט קבוע של משימות.

##### דוגמה: חלוקה סטטית

- **סוכן A:** כל ההזמנות מאירופה
- **סוכן B:** כל ההזמנות מאסיה
- **סוכן C:** כל ההזמנות מאמריקה

**יתרונות:** פשוט, ברור, צפוי

**חסרונות:** לא מאוזן (אם אירופה עמוסה ואסיה פנויה, סוכן A יהיה עמוס ואילו סוכן B יושב בטל)

#### חלוקה דינמית (tnemngissA cimanyD)

משימות מחולקות בזמן אמת על פי זמינות וכושר.

##### נוסחה: חלוקת משימות

$$\text{Task\_Distribution} = \frac{\text{assigned\_tasks}}{\text{max\_tasks}}$$

כאשר:

- **sksat\_dengissa**: מספר המשימות שהוקצו לסוכן
- **sksat\_xam**: מספר המשימות המקסימלי שסוכן יכול לטפל בו במקביל
- סוכן זמין כאשר  $\text{Task\_Distribution} < 1.0$

**יתרונות:** איזון עומסים, ניצול אופטימלי  
**חסרונות:** מורכב יותר ליישום, דורש מעקב בזמן אמת

#### חלוקה מבוססת יכולות (desaB-ytilibapaC)

משימות מחולקות על פי היכולות והמומחיות של כל סוכן.

##### דוגמה: חלוקה מבוססת יכולות

משימה: "אשר הזמנה עם הנחת PIV"

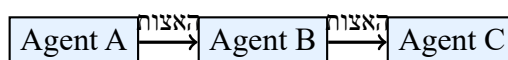
- **סוכן מלאי:** בודק זמינות
- **סוכן תמחור PIV:** מחשב הנחה (בגלל שיש מומחיות ב-PIV)
- **סוכן לוגיסטיקה מהירה:** מתאם משלוח מהיר

## 2.4.6 תיאום זמנים (noitanidrooC)

כאשר מספר סוכנים עובדים על אותה משימה, חשוב לתאם ביניהם [6], [21].

#### laitneuqeS (סידרתי)

בתיאום סידרתי, סוכן A מסיים ומעביר לסוכן B, שמסיים ומעביר לסוכן C, כפי שמוצג באיור 5.6.

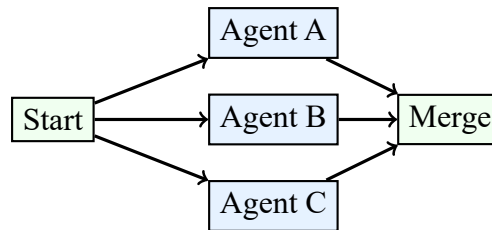


איור 5.6: תיאום סידרתי -- כל סוכן ממתין לקודמו

**יתרונות:** פשוט, ברור, קל לדבג  
**חסרונות:** איטי (כל סוכן חייב לחכות לקודם)

### lellaraP (מקבילי)

בתיאום מקבילי, כמה סוכנים עובדים בו-זמנית על חלקים שונים של המשימה, כפי שמודגם באיור 6.6.

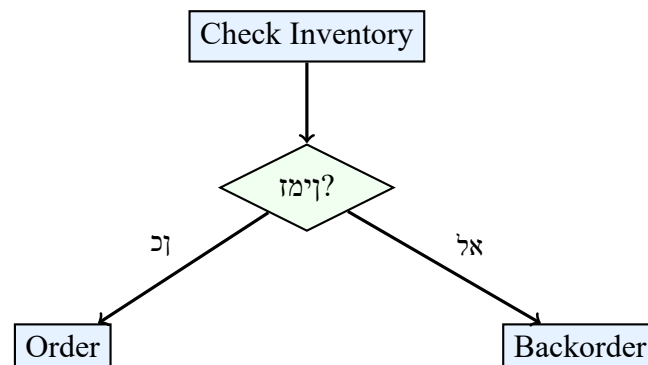


איור 6.6: תיאום מקבילי -- סוכנים עובדים בו-זמנית

**יתרונות:** מהיר, יעיל  
**חסרונות:** דורש סנכרון, מורכב יותר

### lanoitidnoC (מותנה)

בתיאום מותנה, הסוכן הבא תלוי בתוצאת הסוכן הנוכחי. איור 7.6 מציג דוגמה לזרימה כזו.



איור 7.6: תיאום מותנה -- הנתיב נקבע לפי תוצאת הבדיקה

### 3.4.6 עלויות תיאום (daehrevO noitanidrooC)

תיאום בין סוכנים דורש זמן ומשאבים. חשוב למדוד את העלות הזו.

#### נוסחה: עומס תיאום

$$\text{Coordination\_Overhead} = \frac{\text{coordination\_time}}{\text{total\_execution\_time}}$$

כאשר:

□ `emit_noitanidrooc`: הזמן שמוקדש לתקשורת ותיאום בין סוכנים

□ `emit_noitucexe_latot`: סך הזמן לביצוע המשימה

ערך נמוך (> 2.0) מצביע על תיאום יעיל.  
 ערך גבוה (< 5.0) מצביע על עומס תיאום גבוה מדי.

**חישוב עומס תיאום**

משימה: עיבוד הזמנה

□ זמן עבודה אפקטיבי: 03 שניות

□ זמן תקשורת בין סוכנים: 01 שניות

□ סך הזמן: 04 שניות

$$\text{Coordination\_Overhead} = \frac{10}{40} = 0.25 = 25\%$$

זהו עומס סביר. אם התקשורת הייתה לוקחת 52 שניות, היינו מקבלים:

$$\text{Coordination\_Overhead} = \frac{25}{55} = 0.45 = 45\%$$

זהו עומס גבוה מדי, כדאי לשקול מיזוג סוכנים או שיפור התקשורת.

**5.6 טיפול בקונפליקטים**

כאשר מספר סוכנים עובדים יחד, בלתי נמנע שיהיו מצבים שבהם הם לא מסכימים או שפעולותיהם מתנגשות [11], [31].

**1.5.6 סוגי קונפליקטים****קונפליקט משאבים (resource conflict)**

שני סוכנים רוצים להשתמש באותו משאב.

**דוגמה: קונפליקט מלאי**□ **סוכן הזמנה A:** רוצה להקצות את הפריט האחרון במלאי ללקוח PIV□ **סוכן הזמנה B:** רוצה להקצות את אותו פריט להזמנה דחופההפתרון: צריך מנגנון **נעילה (Locking)** או **עדיפויות (Priorities)**.**קונפליקט החלטה (decision conflict)**

שני סוכנים מגיעים להחלטות שונות על אותה בעיה.

**דוגמה: קונפליקט תמחור**□ **סוכן תמחור:** מחשב הנחה של 01%□ **סוכן שיווק:** מציע הנחה של 02% כחלק ממבצע



הפתרון: צריך מנגנון בוררות (Arbitration) או הסלמה (Escalation).

### קונפליקט סדר פעולות (tcilfnoC gniredrO)

שני סוכנים רוצים לעשות פעולות שצריכות להיות בסדר מסוים.

#### דוגמה: קונפליקט עדכון

□ סוכן A: רוצה לעדכן מחיר מוצר

□ סוכן B: רוצה לעדכן מלאי של אותו מוצר

אם שני העדכונים קורים בו-זמנית, עלול להיווצר מצב לא עקבי.  
הפתרון: סנכרון (Synchronization) או עסקאות (Transactions).

## 2.5.6 מנגנוני פתרון קונפליקטים

seitiroirP (עדיפויות)

לכל סוכן יש רמת עדיפות, וסוכן בעדיפות גבוהה יותר "מנצח".

Listing 6.1: Python: Priority-Based Resolution

```

1 class Agent:
2     def __init__(self, name, priority):
3         self.name = name
4         self.priority = priority
5
6 def resolve_conflict(agent1, agent2, resource):
7     """Resolve resource conflict by priority."""
8     if agent1.priority > agent2.priority:
9         print(f"{agent1.name} gets {resource}")
10        return agent1
11    else:
12        print(f"{agent2.name} gets {resource}")
13        return agent2
14
15 # Example
16 vip_agent = Agent("VIP Agent", priority=10)
17 standard_agent = Agent("Standard Agent", priority=5)
18
19 resolve_conflict(vip_agent, standard_agent, "Last item")
20 # Output: VIP Agent gets Last item

```

(SFCF) devreS-tsriF-emoC-tsriF

מי שהגיע ראשון מקבל את המשאב.

gnitoV (הצבעה)

מספר סוכנים "מצביעים" על הפתרון הנכון.

noitalacsE (הסלמה)

אם הסוכנים לא מסכימים, ההחלטה עוברת לסוכן ברמה גבוהה יותר או למנהל אנושי.

Listing 6.2: Python: Escalation Mechanism

```

1 class ConflictManager:
2     def __init__(self):
3         self.escalation_threshold = 3
4         self.conflicts = []
5
6     def handle_conflict(self, agent1, agent2, issue):
7         """Handle conflict between agents."""
8         # Try automated resolution
9         if self.can_resolve_automatically(issue):
10            return self.auto_resolve(agent1, agent2, issue)
11
12        # Escalate to human
13        self.conflicts.append({
14            'agent1': agent1,
15            'agent2': agent2,
16            'issue': issue,
17            'escalated': True
18        })
19
20        print(f"ESCALATION: {issue} needs human decision")
21        return None
22
23    def can_resolve_automatically(self, issue):
24        """Check if issue can be resolved automatically."""
25        return issue.get('auto_resolvable', False)
26
27    def auto_resolve(self, agent1, agent2, issue):
28        """Automatically resolve simple conflicts."""
29        if 'priority' in issue:
30            return agent1 if agent1.priority > agent2.priority else
31            agent2
32        return None

```

## 6.6 noitartsehcrO: תזמור סוכנים

### 1.6.6 מהו noitartsehcrO?

**הגדרה 2.6** (noitartsehcrO). **Orchestration** הוא התהליך של תיאום, ניהול וניטור של מערכת Multi-Agent כדי להבטיח שכל הסוכנים עובדים ביעילות לקראת המטרה המשותפת [51].

תזמור כולל:

- ניתוב הודעות בין סוכנים
- ניהול מצב השיחה
- קביעת סדר ביצוע
- טיפול בשגיאות
- מעקב אחרי התקדמות

## 2.6.6 hparGgnaL: כלי לתזמור

LangGraph הוא ספריית nohtyP המאפשרת לבנות זרימות עבודה (Workflows) מורכבות של סוכנים [7], [8].

### רכיבי hparGgnaL

- sedoN (צמתים): מייצגים סוכנים או פעולות
- segdE (קשתות): מייצגים מעברים בין סוכנים
- etatS (מצב): המידע המשותף בין הסוכנים
- segdE lanoitidnoC (קשתות מותנות): מעברים שתלויים בתנאי

### דוגמה: wolfkroW gnissecorP redrO

Listing 6.3: Python: LangGraph Multi-Agent System

```

1 from langgraph.graph import StateGraph, END
2 from typing import TypedDict, Annotated
3 import operator
4
5 # Define shared state
6 class OrderState(TypedDict):
7     order_id: str
8     customer: str
9     items: list
10    inventory_status: str
11    price: float
12    delivery_date: str
13    payment_status: str
14    errors: Annotated[list, operator.add]
15
16 # Define agent functions
17 def intake_agent(state: OrderState) -> OrderState:
18     """Receive and validate order."""
19     print(f"Intake: Processing order {state['order_id']}")
20     # Validation logic here
21     return state
22
23 def inventory_agent(state: OrderState) -> OrderState:
24     """Check inventory availability."""

```

```

25     print(f"Inventory: Checking stock for {state['items']}")
26     # Check stock
27     state['inventory_status'] = 'available'
28     return state
29
30 def pricing_agent(state: OrderState) -> OrderState:
31     """Calculate final price."""
32     print(f"Pricing: Calculating price")
33     state['price'] = 100.0 # Simplified
34     return state
35
36 def logistics_agent(state: OrderState) -> OrderState:
37     """Schedule delivery."""
38     print(f"Logistics: Scheduling delivery")
39     state['delivery_date'] = '2025-12-20'
40     return state
41
42 def payment_agent(state: OrderState) -> OrderState:
43     """Process payment."""
44     print(f"Payment: Processing ${state['price']}")
45     state['payment_status'] = 'completed'
46     return state
47
48 # Define routing logic
49 def should_continue(state: OrderState) -> str:
50     """Decide next step based on state."""
51     if state.get('inventory_status') != 'available':
52         return 'backorder'
53     return 'continue'
54
55 # Build the graph
56 workflow = StateGraph(OrderState)
57
58 # Add nodes
59 workflow.add_node("intake", intake_agent)
60 workflow.add_node("inventory", inventory_agent)
61 workflow.add_node("pricing", pricing_agent)
62 workflow.add_node("logistics", logistics_agent)
63 workflow.add_node("payment", payment_agent)
64
65 # Add edges
66 workflow.add_edge("intake", "inventory")
67 workflow.add_conditional_edges(
68     "inventory",
69     should_continue,
70     {
71         "continue": "pricing",
72         "backorder": END
73     }
74 )
75 workflow.add_edge("pricing", "logistics")

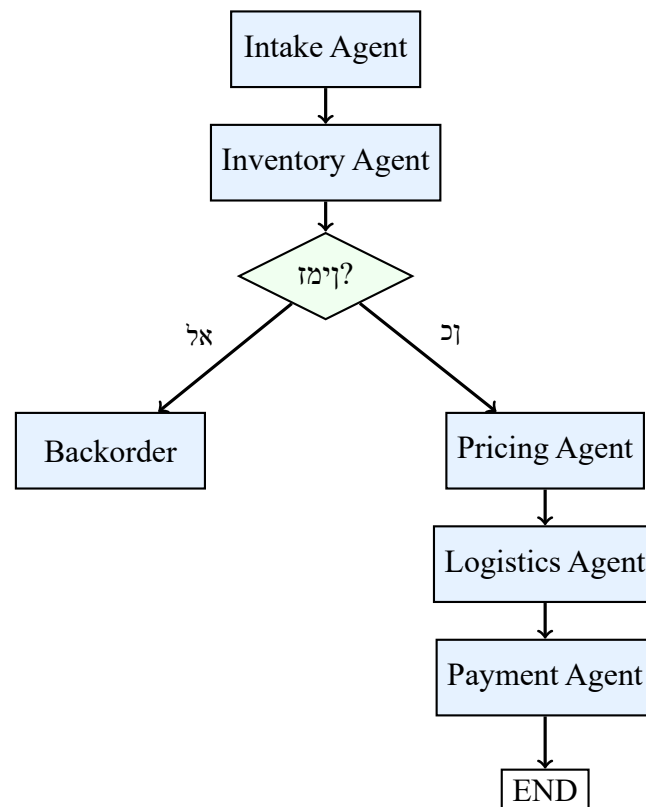
```

```

76 workflow.add_edge("logistics", "payment")
77 workflow.add_edge("payment", END)
78
79 # Set entry point
80 workflow.set_entry_point("intake")
81
82 # Compile
83 app = workflow.compile()
84
85 # Run the workflow
86 initial_state = {
87     "order_id": "ORD-12345",
88     "customer": "John Doe",
89     "items": ["laptop", "mouse"],
90     "errors": []
91 }
92
93 result = app.invoke(initial_state)
94 print(f"\nFinal state: {result}")

```

איור 8.6 מציג את תרשים הזרימה של המערכת.



איור 8.6: זרימת עבודה ב-LangGraph לעיבוד הזמנות

### 3.6.6 etatS tmemeganaM (ניהול מצב)

אחד האתגרים הגדולים ב-Multi-Agent הוא ניהול מצב משותף.

**etatS derahS (מצב משותף)**

כל הסוכנים ניגשים לאותו מצב.

**יתרונות:** פשוט, כולם רואים את אותו מידע

**חסרונות:** קונפליקטים אפשריים, דורש נעילות

**gnissaP egasseM (העברת הודעות)**

כל סוכן שומר את המצב שלו, ושולח הודעות לאחרים.

**יתרונות:** אין קונפליקטים, סוכנים עצמאיים

**חסרונות:** מורכב יותר, קשה יותר לעקוב

**gnicruoS tnevE**

כל שינוי נרשם כאירוע, והמצב הנוכחי הוא סכום כל האירועים.

**יתרונות:** היסטוריה מלאה, ניתן לשחזר מצבים קודמים

**חסרונות:** דורש תשתית נוספת

**7.6 ניטור מערכות tnegA-itluM****1.7.6 מטריקות חשובות**

כדי לנהל מערכת Multi-Agent ביעילות, חשוב למדוד [2], [5]:

1. tuphguorhT **(תפוקה)**: כמה משימות המערכת מעבדת ליחידת זמן

2. ycnetaL **(זמן תגובה)**: כמה זמן לוקח לסיים משימה מקצה לקצה

3. noitazilitU tnegA **(ניצול סוכנים)**: כמה אחוזים מהזמן כל סוכן עובד

4. etaR rorrE **(שיעור שגיאות)**: כמה משימות נכשלות

5. daehrevO noitanidrooC **(עומס תיאום)**: כמה זמן הולך על תקשורת

6. stcillfnoC ecruoseR **(קונפליקטי משאבים)**: כמה קונפליקטים מתרחשים

**2.7.6 כלי ניטור****gniggoL**

תיעוד כל פעולה של כל סוכן.

Listing 6.4: Python: Agent Logging

```

1 import logging
2 from datetime import datetime
3
4 class MonitoredAgent:
5     def __init__(self, name):
6         self.name = name
7         self.logger = logging.getLogger(name)
8         self.logger.setLevel(logging.INFO)

```

```

9
10     # File handler
11     fh = logging.FileHandler(f'{name}_log.txt')
12     formatter = logging.Formatter(
13         '%(asctime)s - %(name)s - %(levelname)s - %(message)s'
14     )
15     fh.setFormatter(formatter)
16     self.logger.addHandler(fh)
17
18     def execute_task(self, task):
19         """Execute task with logging."""
20         self.logger.info(f"Starting task: {task}")
21         start_time = datetime.now()
22
23         try:
24             # Do the work
25             result = self.perform_work(task)
26
27             duration = (datetime.now() - start_time).total_seconds
28
29             self.logger.info(
30                 f"Completed task: {task} in {duration}s"
31             )
32             return result
33
34         except Exception as e:
35             self.logger.error(f"Failed task: {task}, Error: {e}")
36             raise
37
38     def perform_work(self, task):
39         """Actual work simulation."""
40         import time
41         time.sleep(0.5) # Simulate work
42         return f"Result of {task}"
43
44     # Usage
45     agent = MonitoredAgent("InventoryAgent")
46     agent.execute_task("check_stock_item_12345")

```

gnicarT

מעקב אחרי זרימת משימה דרך כל הסוכנים.

Listing 6.5: Python: Simple Tracing

```

1 from typing import Dict, List
2 from dataclasses import dataclass, field
3 from datetime import datetime
4
5 @dataclass
6 class TraceEvent:
7     """Single event in execution trace."""

```

```

8     timestamp: datetime
9     agent: str
10    event_type: str # 'start', 'end', 'message'
11    details: Dict
12
13 @dataclass
14 class ExecutionTrace:
15     """Complete trace of a task execution."""
16     task_id: str
17     events: List[TraceEvent] = field(default_factory=list)
18
19     def add_event(self, agent, event_type, details):
20         """Add event to trace."""
21         event = TraceEvent(
22             timestamp=datetime.now(),
23             agent=agent,
24             event_type=event_type,
25             details=details
26         )
27         self.events.append(event)
28
29     def get_duration(self):
30         """Calculate total execution time."""
31         if len(self.events) < 2:
32             return 0
33         start = self.events[0].timestamp
34         end = self.events[-1].timestamp
35         return (end - start).total_seconds()
36
37     def print_trace(self):
38         """Print execution trace."""
39         print(f"\nTrace for Task {self.task_id}")
40         print("-" * 60)
41         for event in self.events:
42             print(f"{event.timestamp.strftime('%H:%M:%S.%f')}[:-3]}
43 | "
44             f"{event.agent:15} | {event.event_type:10} | "
45             f"{event.details}")
46         print(f"\nTotal Duration: {self.get_duration():.3f}s")
47
48 # Usage
49 trace = ExecutionTrace(task_id="ORD-12345")
50 trace.add_event("IntakeAgent", "start", {"action": "validate_order"
51 })
52 trace.add_event("IntakeAgent", "end", {"status": "valid"})
53 trace.add_event("InventoryAgent", "start", {"action": "check_stock"
54 })
55 trace.add_event("InventoryAgent", "end", {"status": "available"})
56 trace.add_event("PricingAgent", "start", {"action": "calculate"})
57 trace.add_event("PricingAgent", "end", {"price": 100.0})

```



```

56
57 trace.print_trace()

```

draobhsaD scirteM

ויזואליזציה של מצב המערכת בזמן אמת.

#### דשבורד מטריקות

מטריקות שחשוב להציג:

□ sutatS tnegA: איזה סוכנים פעילים/לא פעילים

□ eueuQ ksaT: כמה משימות ממתונות לכל סוכן

□ etaR sseccuS: אחוז ההצלחה של כל סוכן

□ ycnetaL egarevA: זמן ממוצע לכל סוכן

□ hparG noitacinummoC: מי מדבר עם מי

□ goL rorrE: שגיאות אחרונות

## 8.6 דוגמאות מעשיות

### 1.8.6 דוגמה 1: צוות סוכנים לטיפול בהזמנה

נבנה מערכת שלמה לטיפול בהזמנות מקצה לקצה.

Listing 6.6: Python: Complete Order Processing System

```

1 from typing import Dict, Any
2 from dataclasses import dataclass
3 from enum import Enum
4
5 class OrderStatus(Enum):
6     NEW = "new"
7     VALIDATED = "validated"
8     IN_STOCK = "in_stock"
9     OUT_OF_STOCK = "out_of_stock"
10    PRICED = "priced"
11    SCHEDULED = "scheduled"
12    PAID = "paid"
13    COMPLETED = "completed"
14    FAILED = "failed"
15
16 @dataclass
17 class Order:
18     order_id: str
19     customer: str
20     items: list

```

```

21     status: OrderStatus = OrderStatus.NEW
22     price: float = 0.0
23     delivery_date: str = ""
24     payment_id: str = ""
25     error: str = ""
26
27 class OrderProcessingSystem:
28     def __init__(self):
29         self.agents = {
30             'intake': IntakeAgent(),
31             'inventory': InventoryAgent(),
32             'pricing': PricingAgent(),
33             'logistics': LogisticsAgent(),
34             'payment': PaymentAgent()
35         }
36
37     def process_order(self, order: Order) -> Order:
38         """Process order through all agents."""
39         print(f"\n{'='*60}")
40         print(f"Processing Order: {order.order_id}")
41         print(f"{'='*60}")
42
43         # Step 1: Intake
44         order = self.agents['intake'].process(order)
45         if order.status == OrderStatus.FAILED:
46             return order
47
48         # Step 2: Inventory
49         order = self.agents['inventory'].process(order)
50         if order.status == OrderStatus.OUT_OF_STOCK:
51             print("Order on backorder")
52             return order
53
54         # Step 3: Pricing
55         order = self.agents['pricing'].process(order)
56
57         # Step 4: Logistics
58         order = self.agents['logistics'].process(order)
59
60         # Step 5: Payment
61         order = self.agents['payment'].process(order)
62
63         if order.status == OrderStatus.PAID:
64             order.status = OrderStatus.COMPLETED
65             print(f"\n*** Order {order.order_id} COMPLETED ***")
66
67         return order
68
69 class IntakeAgent:
70     def process(self, order: Order) -> Order:
71         """Validate order data."""

```

```

72     print(f"\n[INTAKE] Validating order {order.order_id}")
73
74     if not order.customer:
75         order.status = OrderStatus.FAILED
76         order.error = "Missing customer info"
77         return order
78
79     if not order.items:
80         order.status = OrderStatus.FAILED
81         order.error = "No items in order"
82         return order
83
84     order.status = OrderStatus.VALIDATED
85     print(f"[INTAKE] Order validated: {len(order.items)} items")
86 )
87
88     return order
89
90 class InventoryAgent:
91     def __init__(self):
92         self.stock = {
93             'laptop': 10,
94             'mouse': 50,
95             'keyboard': 30
96         }
97
98     def process(self, order: Order) -> Order:
99         """Check inventory."""
100         print(f"\n[INVENTORY] Checking stock")
101
102         for item in order.items:
103             if item not in self.stock:
104                 print(f"[INVENTORY] Item not found: {item}")
105                 order.status = OrderStatus.OUT_OF_STOCK
106                 return order
107
108             if self.stock[item] < 1:
109                 print(f"[INVENTORY] Out of stock: {item}")
110                 order.status = OrderStatus.OUT_OF_STOCK
111                 return order
112
113         order.status = OrderStatus.IN_STOCK
114         print(f"[INVENTORY] All items available")
115         return order
116
117 class PricingAgent:
118     def __init__(self):
119         self.prices = {
120             'laptop': 1000.0,
121             'mouse': 25.0,
122             'keyboard': 75.0
123         }

```

```

122     def process(self, order: Order) -> Order:
123         """Calculate price."""
124         print(f"\n[PRICING] Calculating price")
125
126         total = sum(self.prices.get(item, 0) for item in order.
127 items)
128
129         # VIP discount
130         if 'VIP' in order.customer:
131             total *= 0.9
132             print(f"[PRICING] VIP discount applied")
133
134         order.price = total
135         order.status = OrderStatus.PRICED
136         print(f"[PRICING] Total: ${total:.2f}")
137         return order
138
139     class LogisticsAgent:
140         def process(self, order: Order) -> Order:
141             """Schedule delivery."""
142             print(f"\n[LOGISTICS] Scheduling delivery")
143
144             # Simple scheduling
145             order.delivery_date = "2025-12-20"
146             order.status = OrderStatus.SCHEDULED
147             print(f"[LOGISTICS] Delivery: {order.delivery_date}")
148             return order
149
150     class PaymentAgent:
151         def process(self, order: Order) -> Order:
152             """Process payment."""
153             print(f"\n[PAYMENT] Processing payment of ${order.price:.2f}
154             ")
155
156             # Simulate payment
157             order.payment_id = f"PAY-{order.order_id}"
158             order.status = OrderStatus.PAID
159             print(f"[PAYMENT] Payment successful: {order.payment_id}")
160             return order
161
162     # Run example
163     system = OrderProcessingSystem()
164
165     order1 = Order(
166         order_id="ORD-001",
167         customer="John Doe VIP",
168         items=['laptop', 'mouse']
169     )
170
171     result = system.process_order(order1)

```

```

171 print(f"\nFinal Status: {result.status.value}")
172 print(f"Price: ${result.price:.2f}")
173 print(f"Delivery: {result.delivery_date}")

```

## 2.8.6 דוגמה 2: סוכן מכירות + סוכן מלאי + סוכן לוגיסטיקה

דוגמה לתקשורת ישירה בין שלושה סוכנים בארכיטקטורת Mesh.

Listing 6.7: Python: Multi-Agent Mesh Communication

```

1 from typing import Optional, Dict
2
3 class Message:
4     def __init__(self, from_agent: str, to_agent: str,
5                 msg_type: str, content: Dict):
6         self.from_agent = from_agent
7         self.to_agent = to_agent
8         self.msg_type = msg_type
9         self.content = content
10
11 class Agent:
12     def __init__(self, name: str):
13         self.name = name
14         self.inbox = []
15
16     def send_message(self, to_agent: 'Agent',
17                    msg_type: str, content: Dict):
18         """Send message to another agent."""
19         msg = Message(self.name, to_agent.name, msg_type, content)
20         to_agent.receive_message(msg)
21         print(f"[{self.name}] -> [{to_agent.name}]: {msg_type}")
22
23     def receive_message(self, msg: Message):
24         """Receive message from another agent."""
25         self.inbox.append(msg)
26
27     def process_messages(self):
28         """Process all pending messages."""
29         while self.inbox:
30             msg = self.inbox.pop(0)
31             self.handle_message(msg)
32
33     def handle_message(self, msg: Message):
34         """Handle incoming message - to be overridden."""
35         pass
36
37 class SalesAgent(Agent):
38     def __init__(self):
39         super().__init__("SalesAgent")
40         self.current_order = None
41
42     def create_order(self, items: list,

```

```

43         inventory_agent: Agent, logistics_agent: Agent)
44     :
45     """Create new order and coordinate with other agents."""
46     self.current_order = {
47         'order_id': 'ORD-123',
48         'items': items,
49         'status': 'pending'
50     }
51
52     print(f"\n[{self.name}] Creating order: {items}")
53
54     # Ask inventory about availability
55     self.send_message(
56         inventory_agent,
57         'check_availability',
58         {'items': items, 'order_id': 'ORD-123'}
59     )
60
61     def handle_message(self, msg: Message):
62         """Handle responses from other agents."""
63         if msg.msg_type == 'availability_confirmed':
64             print(f"[{self.name}] Stock confirmed!")
65             # Can now proceed with pricing, etc.
66
67         elif msg.msg_type == 'availability_denied':
68             print(f"[{self.name}] Out of stock: {msg.content}")
69             self.current_order['status'] = 'backorder'
70
71     class InventoryAgent(Agent):
72         def __init__(self):
73             super().__init__("InventoryAgent")
74             self.stock = {'laptop': 5, 'mouse': 20}
75
76         def handle_message(self, msg: Message):
77             """Handle inventory requests."""
78             if msg.msg_type == 'check_availability':
79                 items = msg.content['items']
80                 all_available = all(
81                     self.stock.get(item, 0) > 0 for item in items
82                 )
83
84                 if all_available:
85                     print(f"[{self.name}] All items available")
86                     # Respond to sales agent
87                     # (in real system would use reference to sender)
88                 else:
89                     print(f"[{self.name}] Some items unavailable")
90
91     class LogisticsAgent(Agent):
92         def __init__(self):
93             super().__init__("LogisticsAgent")

```

```

93
94     def handle_message(self, msg: Message):
95         """Handle logistics requests."""
96         if msg.msg_type == 'schedule_delivery':
97             print(f"[{self.name}] Scheduling delivery")
98
99     # Create agents
100     sales = SalesAgent()
101     inventory = InventoryAgent()
102     logistics = LogisticsAgent()
103
104     # Process order
105     sales.create_order(['laptop', 'mouse'], inventory, logistics)
106     inventory.process_messages()

```

### 3.8.6 דוגמה 3: מערכת tnegA-itluM לתמיכת לקוחות

מערכת עם סוכני תמיכה מרובים שמטפלים בפניות שונות בו-זמנית.

#### מערכת תמיכה

##### סוכנים:

- tnegA retuoR: מנתב פניות לסוכן המתאים
- tnegA troppuS lacinhcet: תמיכה טכנית
- tnegA gnilliB: שאלות חיוב
- tnegA troppuS lareneG: שאלות כלליות
- tnegA noitalacsE: טיפול בבעיות מורכבות

##### זרימה:

1. לקוח שולח שאלה
2. retuoR מנתח את השאלה וקובע לאיזה סוכן לשלוח
3. הסוכן המתאים מטפל בפנייה
4. אם הבעיה מורכבת מדי, הסוכן מעביר ל-tnegA noitalacsE
5. tnegA noitalacsE יכול לשתף פעולה עם מספר סוכנים או להעביר לאדם

## 9.6 תרגילים

## 1.9.6 תרגיל 1: תכנון מערכת tnegA-itluM

## תכנון מערכת tnegA-itluM

**תרחיש:** בנק רוצה לבנות מערכת לאישור הלוואות באופן אוטומטי.  
**משימה:**

1. זהה את הסוכנים הנדרשים (לפחות 4)
2. בחר ארכיטקטורה מתאימה (lacihcrareIH / hseM / ekopS-dna-buH)
3. צייר תרשים של הארכיטקטורה
4. הגדר את התקשורת בין הסוכנים
5. זהה קונפליקטים אפשריים והצע פתרון

**פתרון מוצע:**

## 1. סוכנים נדרשים:

□ tnegA noitacilppA: מקבל את הבקשה, מוודא שהיא מלאה

□ tnegA erocS tiderC: בודק ציון אשראי של הלקוח

□ tnegA noitacifireV emocnI: מאמת הכנסות

□ tnegA tmemssessA ksiR: מעריך סיכון

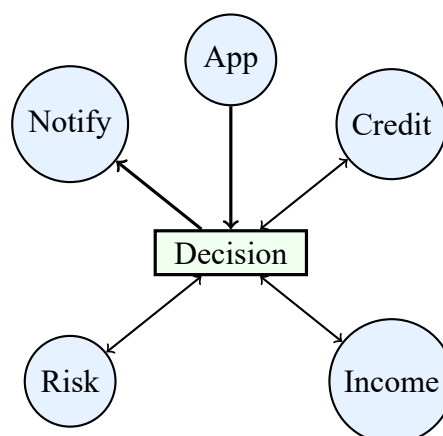
□ tnegA noisiced: מחליט האם לאשר או לדחות

□ tnegA noitacifitoN: שולח הודעה ללקוח

## 2. ארכיטקטורה: Hub-and-Spoke עם Decision Agent כ-Hub.

**סיבה:** ההחלטה צריכה להתקבל במקום אחד, על בסיס כל הנתונים. Hub-and-Spoke מבטיח שכל המידע עובר דרך סוכן החלטה אחד.

## 3. תרשים: איור 9.6 מציג את הארכיטקטורה המוצעת.



איור 9.6: מערכת אישור הלוואות בארכיטקטורת Hub-and-Spoke

## 4. תקשורת:



□  $\text{tnegA noisiceD} \rightarrow \text{tnegA noitacilppA}$ : נתוני בקשה

□  $\text{tnegA noisiceD} \leftrightarrow \text{ksiR/emocnI/tiderC}$ : בקשות מידע

□  $\text{tnegA noisiceD} \rightarrow \text{tnegA noitacifitoN}$ : החלטה סופית

## 5. קונפליקטים אפשריים:

□ **קונפליקט**:  $\text{erocS tiderC}$  אומר "אשר", אבל  $\text{tnemssessA ksiR}$  אומר "דחה"

□ **פתרון**:  $\text{tnegA noisiceD}$  משקלל את כל הגורמים לפי משקלות מוגדרים מראש (למשל:  $\text{tiderC}$  04%,  $\text{emocnI}$  03%,  $\text{ksiR}$  03%)

## 2.9.6 תרגיל 2: זיהוי נקודות כשל

### זיהוי נקודות כשל

נתונה מערכת Multi-Agent בארכיטקטורת Hub-and-Spoke:

□  $\text{buH} : \text{tnegA rotartsehcrO}$

□  $\text{stnega rekrow 5} : \text{sekoPS}$

### שאלות:

1. מה קורה אם ה- $\text{rotartsehcrO}$  נכשל?
2. מה קורה אם אחד מה- $\text{srekroW}$  נכשל?
3. איך אפשר להפוך את המערכת לחסינה יותר?

### פתרון:

1. אם  $\text{rotartsehcrO}$  נכשל:

□ כל המערכת נופלת -- אף  $\text{rekroW}$  לא יכול לקבל משימות

□ זוהי נקודת כשל יחידה קריטית

2. אם  $\text{rekroW}$  נכשל:

□ המערכת ממשיכה לעבוד

□ המשימות של ה- $\text{rekroW}$  הנכשל לא יבוצעו

□  $\text{rotartsehcrO}$  צריך לזהות ולנתב מחדש

3. הפיכת המערכת לחסינה יותר:

**פתרון א:**  $\text{rotaRtnadnecy}$

□ הרץ שני  $\text{rotaRtnadnecy}$  במצב Active-Passive

□ אם הראשי נכשל, הגיבוי נכנס לפעולה

**פתרון ב:**  $\text{skcehC htlaeH rekroW}$

- rotartsehcrO בודק כל 01 שניות שכל ה-srekroW חיים
- אם rekroW לא עונה, rotartsehcrO מפסיק לשלוח לו משימות
- כשה-rekroW חוזר, הוא מצטרף בחזרה

**פתרון ג: msinahceM yrteR ksaT**

- אם rekroW לא מחזיר תשובה תוך זמן מוגדר
- rotartsehcrO שולח את המשימה ל-rekroW אחר

### 3.9.6 תרגיל 3: בניית מדיניות noitalacsE

#### מדיניות noitalacsE

בנה מדיניות Escalation למערכת תמיכת לקוחות עם 3 רמות:

1. troppuS 1L: סוכן אוטומטי
2. troppuS 2L: סוכן מתקדם
3. troppuS 3L: אדם (מומחה)

הגדר:

- מתי לעבור מ-1L ל-2L?
- מתי לעבור מ-2L ל-3L?
- מהם הקריטריונים?

**פתרון מוצע:**

**מעבר מ-1L ל-2L:**

1. erocS ecnedifnoC **נמוך**: אם 1L לא בטוח בתשובה ( $ecnedifnoc > 7.0$ )
2. **מספר ניסיונות**: אם 1L ניסה 3 פעמים ולא הצליח לפתור
3. **בקשה מפורשת**: אם הלקוח מבקש "לדבר עם מישו אחר"
4. **נושא מורכב**: אם הנושא בקטגוריה מוגדרת כמורכבת

**מעבר מ-2L ל-3L:**

1. **בעיה קריטית**: תקלה במערכת שמשפיעה על לקוחות רבים
2. **ערך גבוה**: לקוח PIV עם בעיה שלא נפתרה ב-2L
3. **זמן ארוך**: בעיה פתוחה מעל 42 שעות
4. **מגבלות טכניות**: 2L לא יכול לבצע פעולה (למשל: החזר כספי מעל סכום מסוים)

**קוד לדוגמה:**

Listing 6.8: Python: Escalation Policy

```

1 from dataclasses import dataclass
2 from typing import Optional
3 from datetime import datetime, timedelta
4
5 @dataclass
6 class SupportTicket:
7     ticket_id: str
8     customer: str
9     issue: str
10    level: int = 1
11    attempts: int = 0
12    created_at: datetime = None
13    is_vip: bool = False
14
15    def __post_init__(self):
16        if self.created_at is None:
17            self.created_at = datetime.now()
18
19 class EscalationPolicy:
20     def should_escalate_to_l2(self, ticket: SupportTicket,
21                             confidence: float) -> bool:
22         """Check if should escalate from L1 to L2."""
23         # Low confidence
24         if confidence < 0.7:
25             return True
26
27         # Too many attempts
28         if ticket.attempts >= 3:
29             return True
30
31         # Complex category
32         complex_keywords = ['refund', 'legal', 'security breach']
33         if any(kw in ticket.issue.lower() for kw in
34               complex_keywords):
35             return True
36
37         return False
38
39     def should_escalate_to_l3(self, ticket: SupportTicket) -> bool:
40         """Check if should escalate from L2 to L3."""
41         # VIP customer
42         if ticket.is_vip:
43             return True
44
45         # Open too long
46         time_open = datetime.now() - ticket.created_at
47         if time_open > timedelta(hours=24):
48             return True
49
50         # Critical issue

```

```

50 |     critical_keywords = ['system down', 'data loss', 'security'
51 | ]
52 |     if any(kw in ticket.issue.lower() for kw in
critical_keywords):
53 |         return True
54 |
55 |     return False
56 |
57 |     def escalate(self, ticket: SupportTicket,
58 |                 confidence: Optional[float] = None):
59 |         """Escalate ticket to appropriate level."""
60 |         if ticket.level == 1:
61 |             if self.should_escalate_to_l2(ticket, confidence or
1.0):
62 |                 ticket.level = 2
63 |                 print(f"ESCALATED to L2: Ticket {ticket.ticket_id}")
64 |             )
65 |
66 |             elif ticket.level == 2:
67 |                 if self.should_escalate_to_l3(ticket):
68 |                     ticket.level = 3
69 |                     print(f"ESCALATED to L3: Ticket {ticket.ticket_id}")
70 |                 )
71 |
72 | # Example
73 | policy = EscalationPolicy()
74 |
75 | ticket1 = SupportTicket(
76 |     ticket_id="T001",
77 |     customer="John Doe",
78 |     issue="Need refund for order",
79 |     attempts=1
80 | )
81 |
82 | policy.escalate(ticket1, confidence=0.5)
83 | # Output: ESCALATED to L2: Ticket T001

```

## 4.9.6 תרגיל 4: תכנון ניטור

## תכנון ניטור

תכנון מערכת ניטור למערכת Multi-Agent עם 01 סוכנים.  
הגדר:

1. אלו מטריקות לאסוף?

2. באיזו תדירות?

3. מהם ה-Thresholds (ספים) לאזהרות?

4. איך להציג את המידע?

פתרון:

1. מטריקות לאסוף:

ברמת סוכן בודד:

□ sutatS :nwod/pu

□ sksaT :detelpmoC מספר משימות שהושלמו

□ sksaT :deliaF מספר משימות שנכשלו

□ egarevA :ycnetaL זמן ממוצע למשימה

□ tnerruC :daoL כמה משימות פעילות כרגע

□ yromeM/UPC :egasU ניצול משאבים

ברמת המערכת:

□ latoT :tuphguorhT משימות לשנייה של כל המערכת

□ egarevA :ycnetaL dnE-ot-dnE זמן מקצה לקצה

□ evitcA :stnegA כמה סוכנים פעילים

□ egasseM :eziS eueuQ כמה הודעות בתור

□ noitanidrooC :daehrevO עומס תיאום כולל

2. תדירות איסוף:

□ emit-laeR :scirtem (כל שנייה): ,sutatS ,tnerruC daoL

□ tsaF :scirtem (כל 01 שניות): ,ycnetaL ,tuphguorhT

□ wols :scirtem (כל דקה): ,yromeM/UPC ,ksaT stnuoc

3. sdlohserhT לאזהרות:

הקירטמ	הרהזא	יטירק
Agent Status	-	Down
Error Rate	> 5%	> 10%
Average Latency	> 2s	> 5s
Agent Load	> 80%	> 95%
Queue Size	> 100	> 500
Coordination Overhead	> 40%	> 60%

#### 4. הצגת מידע: draobhsaD ראשי:

- lenaP weivrevO: סך סוכנים פעילים, tuphguorht כולל, etar rorre כולל
- dirG tnegA: טבלה עם שורה לכל סוכן, עמודות: ,srrorE ,ycnetaL ,daoL ,sutatS
- krowteN hparG: גרף המראה תקשורת בין סוכנים (עובי קו = כמות הודעות)
- trahC enilemiT: ycnetaL לאורך זמן
- lenaP trelA: רשימת התראות אקטיביות

### 5.9.6 תרגיל 5: ניתוח עלויות מול יעילות

#### ניתוח עלויות

חברה שוקלת בין שתי אפשרויות:  
**אפשרות A:** סוכן יחיד גדול

- עלות פיתוח: \$000,05
- עלות תפעול חודשית: \$000,1
- זמן עיבוד משימה: 01 שניות
- יכולת: 001 משימות/דקה
- אפשרות B:** 5 סוכנים קטנים
- עלות פיתוח: \$000,08 (5 □ \$000,61)
- עלות תפעול חודשית: \$005,1
- זמן עיבוד משימה: 5 שניות (כל סוכן)
- יכולת: 002 משימות/דקה (במקביל)

#### שאלות:

1. מהי נקודת האיזון (neve-kaerb)?
2. איזו אפשרות עדיפה לטווח ארוך (3 שנים)?
3. מהן העלויות הנוספות שצריך לקחת בחשבון?

#### פתרון:

#### 1. עלויות כוללות: אפשרות A:

$$\text{Total Cost (t months)} = 50,000 + 1,000t$$

#### אפשרות B:

$$\text{Total Cost (t months)} = 80,000 + 1,500t$$

### נקודת איזון:

$$50,000 + 1,000t = 80,000 + 1,500t$$

$$-500t = 30,000$$

$$t = -60$$

מכיוון ש- $t$  שלילי, אין נקודת איזון -- אפשרות A תמיד זולה יותר מבחינה כלכלית נטו! אבל, זה לא כל הסיפור...

2. ניתוח ערך (3 שנים = 63 חודשים):

אפשרות A:

$$\text{Cost}_A = 50,000 + 1,000 \times 36 = 86,000$$

תפוקה: 001 משימות/דקה

אפשרות B:

$$\text{Cost}_B = 80,000 + 1,500 \times 36 = 134,000$$

תפוקה: 002 משימות/דקה

עלות למשימה:

נניח 1,000,000 משימות בשנה:

אפשרות A:

$$\text{Cost per task} = \frac{86,000}{3,000,000} = \$0.0287$$

אפשרות B:

$$\text{Cost per task} = \frac{134,000}{3,000,000} = \$0.0447$$

נראה שאפשרות A זולה יותר!

אבל, מה אם הביקוש גדל?

אם מגיעות 003 משימות/דקה:

□ אפשרות A: לא מספיקה (רק 001/דקה) -- צריך להוסיף סוכנים

□ אפשרות B: עדיין מספיקה (002/דקה)

אם צריך להכפיל את A:

$$\text{Cost}_A = 2 \times 86,000 = 172,000$$

עכשיו B זולה יותר!

3. עלויות נוספות:

□ תחזוקה: tnegA-itluM מורכב יותר לתחזוקה (+02%)

□ ניטור: צריך כלים לניטור tnegA-itluM (+005\$/חודש)

□ תקלות: אם סוכן בודד נופל, כל המערכת נופלת (עלות emitnwod)

□ **גמישות:** tnegA-itluM מאפשר שדרוגים חלקיים (חיסכון עתידי)

□ **סקלביליות:** tnegA-itluM קל יותר להרחבה

**המלצה:**

□ אם הביקוש צפוי ויציב (> 001 משימות/דקה): אפשרות A

□ אם הביקוש משתנה או צפוי לצמוח: אפשרות B

□ אם חשוב זמינות גבוהה (emitpu): אפשרות B

## 6.9.6 תרגיל 6: תקשורת בין שני סוכנים (nohtyP)

### nohtyP: תקשורת בסיסית

בנה מערכת תקשורת פשוטה בין שני סוכנים:

□ tnegA A: שואל שאלה

□ tnegA B: עונה על השאלה

יש ליישם:

1. מחלקת Message

2. מחלקת Agent בסיסית

3. מנגנון שליחה וקבלה

4. לוג של כל ההודעות

**פתרון:**

Listing 6.9: Python: Basic A2A Communication

```

1 from typing import Dict, List, Optional
2 from dataclasses import dataclass, field
3 from datetime import datetime
4 import json
5
6 @dataclass
7 class Message:
8     """A2A message."""
9     msg_id: str
10    from_agent: str
11    to_agent: str
12    msg_type: str
13    payload: Dict
14    timestamp: datetime = field(default_factory=datetime.now)
15
16    def to_json(self) -> str:
17        """Convert message to JSON."""
18        return json.dumps({
19            'msg_id': self.msg_id,
```



```

20         'from': self.from_agent,
21         'to': self.to_agent,
22         'type': self.msg_type,
23         'payload': self.payload,
24         'timestamp': self.timestamp.isoformat()
25     }, indent=2)
26
27 class MessageBroker:
28     """Central message broker for A2A communication."""
29     def __init__(self):
30         self.agents: Dict[str, 'Agent'] = {}
31         self.message_log: List[Message] = []
32
33     def register_agent(self, agent: 'Agent'):
34         """Register agent with broker."""
35         self.agents[agent.name] = agent
36         print(f"[BROKER] Registered: {agent.name}")
37
38     def send_message(self, msg: Message):
39         """Route message to recipient."""
40         self.message_log.append(msg)
41
42         print(f"\n[BROKER] Routing message:")
43         print(f"  From: {msg.from_agent}")
44         print(f"  To: {msg.to_agent}")
45         print(f"  Type: {msg.msg_type}")
46
47         if msg.to_agent in self.agents:
48             self.agents[msg.to_agent].receive_message(msg)
49         else:
50             print(f"[BROKER] ERROR: Agent {msg.to_agent} not found")
51
52     def print_log(self):
53         """Print all messages."""
54         print(f"\n{'='*60}")
55         print("MESSAGE LOG")
56         print(f"{'='*60}")
57         for i, msg in enumerate(self.message_log, 1):
58             print(f"\n--- Message {i} ---")
59             print(msg.to_json())
60
61 class Agent:
62     """Basic agent with A2A capabilities."""
63     def __init__(self, name: str, broker: MessageBroker):
64         self.name = name
65         self.broker = broker
66         self.inbox: List[Message] = []
67         self.msg_counter = 0
68
69         # Register with broker

```

```

70     broker.register_agent(self)
71
72     def send(self, to_agent: str, msg_type: str, payload: Dict):
73         """Send message to another agent."""
74         self.msg_counter += 1
75         msg = Message(
76             msg_id=f"{self.name}-{self.msg_counter}",
77             from_agent=self.name,
78             to_agent=to_agent,
79             msg_type=msg_type,
80             payload=payload
81         )
82
83         print(f"\n[{self.name}] Sending: {msg_type} to {to_agent}")
84         self.broker.send_message(msg)
85
86     def receive_message(self, msg: Message):
87         """Receive message."""
88         self.inbox.append(msg)
89         print(f"[{self.name}] Received: {msg.msg_type}")
90         self.handle_message(msg)
91
92     def handle_message(self, msg: Message):
93         """Handle incoming message - to be overridden."""
94         print(f"[{self.name}] Processing: {msg.msg_type}")
95
96 class QuestionAgent(Agent):
97     """Agent that asks questions."""
98     def ask_question(self, to_agent: str, question: str):
99         """Ask a question."""
100         self.send(
101             to_agent,
102             'question',
103             {'question': question}
104         )
105
106     def handle_message(self, msg: Message):
107         """Handle answer."""
108         if msg.msg_type == 'answer':
109             answer = msg.payload.get('answer')
110             print(f"[{self.name}] Got answer: {answer}")
111
112 class AnswerAgent(Agent):
113     """Agent that answers questions."""
114     def __init__(self, name: str, broker: MessageBroker):
115         super().__init__(name, broker)
116         self.knowledge = {
117             'What is AI?': 'Artificial Intelligence',
118             'What is A2A?': 'Agent-to-Agent communication',
119             'What is Python?': 'A programming language'
120         }

```

```

121
122     def handle_message(self, msg: Message):
123         """Handle question and send answer."""
124         if msg.msg_type == 'question':
125             question = msg.payload.get('question')
126             answer = self.knowledge.get(question, 'I don\'t know')
127
128             print(f"[{self.name}] Question: {question}")
129             print(f"[{self.name}] Answer: {answer}")
130
131             # Send answer back
132             self.send(
133                 msg.from_agent,
134                 'answer',
135                 {'answer': answer, 'original_question': question}
136             )
137
138     # Example usage
139     print("="*60)
140     print("A2A COMMUNICATION DEMO")
141     print("="*60)
142
143     # Create broker
144     broker = MessageBroker()
145
146     # Create agents
147     alice = QuestionAgent("Alice", broker)
148     bob = AnswerAgent("Bob", broker)
149
150     # Alice asks questions
151     alice.ask_question("Bob", "What is AI?")
152     alice.ask_question("Bob", "What is A2A?")
153
154     # Print message log
155     broker.print_log()

```

## 01.6 סיכום

בפרק זה למדנו:

- פרוטוקול A2A: תקשורת בין סוכנים אוטונומיים
- ארכיטקטורות: lacihcrareiH ,hseM ,ekopS-dna-buH
- תיאום משימות: חלוקה סטטית, דינמית ומבוססת יכולות
- קונפליקטים: זיהוי וטיפול במצבי התנגשות
- תזמור: שימוש ב-hparGgnaL לניהול זרימות עבודה
- ניטור: מטריקות וכלים למעקב אחרי tnegA-itluM

□ **דוגמאות מעשיות:** מערכות לעיבוד הזמנות ותמיכת לקוחות

### 1.01.6 נקודות מפתח למנהלים

1. tnegA-itluM לא תמיד הפתרון: לפעמים סוכן יחיד פשוט יותר וזול יותר
2. תכנון הארכיטקטורה קריטי: בחירת ארכיטקטורה שגויה עלולה ליצור בעיות תחזוקה
3. תיאום עולה כסף: daehrevO noitanidrooC יכול להיות 02%-05% מהזמן
4. טיפול בקונפליקטים הכרחי: צריך מנגנונים ברורים לפתרון סכסוכים
5. ניטור הוא evah-tsum: ללא ניטור, בלתי אפשרי לנהל tnegA-itluM בייצור
6. התחל קטן: עדיף להתחיל עם 2-3 סוכנים ולהוסיף בהדרגה

### 2.01.6 המשך

בפרק הבא נלמד על GAR (noitareneG detnemguA-laveirteR) -- טכניקה המאפשרת לסוכנים לגשת למידע עדכני ומדויק מחוץ ל-MLL, דרך אחזור ממאגרי ידע.

## מקורות

- [1] <https://a2a-protocol.org/latest/specification/>, (0.1v TFARD) noitacificepS locotorP (A2A) tnegA2tnegA ,tcejorP A2A gnisu noitacificeps locotorp laiciffO ,PTTH rev0 0.2 CPR-NOSJ .5202 ,(S)
- [2] tnegA-itluM gnivlovE nA :sisongaiD ot ataD ytilibavresbO morF`` ,la te ,uiL .W ,nehC .F rof tnegAspO ,5202 ,tnirperp viXra ,smetsyS duolC ni tmemeganaM tnedicnI rof metsyS .2510.24145 :viXra .secart dna ,sgol ,scirtem gnisu tmemeganaM tnedicni duolc
- [3] rof krowemarF tnegA-itluM lacihrareiH A :artsehcrOtnegA`` ,la te ,gnaW .J ,nehC .L lartnec htiw krowemarF lacihrareiH ,5202 ,tnirperp viXra ,gnivloS ksaT esopruP-lareneG .2506.12508 :viXra .sksat-bus setageled dna sevitcejbo sesopmoted taht tnegA gninnalp
- [4] <https://developers.googleblog.com/en/a2a-a-new-era-of-agent-interopability/> , (A2A) locotorP tnegA2tnegA eht gnicnuonna ,elgooG .5202 ,ytilibareporetni tnegA rof locotorp A2A fo tmemecnuonna laiciffO
- [5] dna ,scitylanA ,ytilibavresbO :gnikramhcneB xOB-kcalB dnoyeb`` ,la te ,eel .S ,nosnhoJ .D ni segnellahc euqinu sesserdda ,5202 ,tnirperp viXra ,smetsyS citnegA fo noitazimitpO .2503.06745 :viXra .snoitacilppa desab-MLL gnirotinom
- [6] ,GAD cimanyD htiw noitanidrooc tnegA-itluM gnicnahnE :CAMEd`` ,la te ,kraP .J ,miK .S sesserdda ,5202 PLNME :scitsiugniL lanoitatupmoC rof noitaicossA eht fo sgnidniF ni lanoitatupmoC rof noitaicossA ,sessecorp tnegA-itlum ni ycnedneped dna ytniatrecnu ksat .5202 ,scitsiugniL
- [7] <https://blog.langchain.com/langgraph-multi-agent-workflows/> ,swolfkroW tnegA-itluM :hparGnaL ,niahCgnaL tnegA-itlum no ediug laiciffO ,com/langgraph-multi-agent-workflows/ .4202 ,noitcurtsnoc wolfkrow
- [8] <https://www.langchain.com/langgraph> ,krowemarF noitartsehcrO tnegA level-woL :hparGnaL ,niahCgnaL -itlum rof hparGetatS htiw noitartsehcro desab-GAD ,langchain.com/langgraph .5202 ,swolfkrow tnegA
- [9] gnisu krowemarF lanoitasrevnoC tnegA-itluM A :retsaMtnegA`` ,la te ,gnahZ .K ,iL .M ,tnirperp viXra ,sisylanA dna laveirteR noitamrofni ladomitluM rof sloctorP PCM dna A2A :viXra .noitanidrooc tnegA-itlum rof sloctorp PCM dna A2A gninibmoc krowemarF ,5202 .2507.21105
- [01] <https://www.linuxfoundation.org/press/linux-foundation-launches-the-agent2agent-protocol-project-to-enable-secure-intelligent-communication-between-ai-agents> ,tcejorP locotorP tnegA2tnegA eht sehcnuaL noitadnuoF xuniL ,noitadnuoF xuniL rednu ecnanrevog A2A ,communication-between-ai-agents .5202 ,snoitazinagro gnitroppus +051 htiw

- [11] "weiveR lacinheT :smetsyS tnegA-itluM ni seuqinhceT noituloseR tcilfnoC` ,la te htimS .J ,stcilfnoc laog otni noituloser tcilfnoc sezirogetaC ,5202 ,lanruoJ seidutS nredoM naeporuE .gnigrem feileb dna ,stcilfnoc noitamrofni
- [21] -noisiceD evitarepooC tnegA-itluM no yevruS evisneherpmoC A` ,la te ,gnahZ .Y ,gnaW .L noitulovE ,5202 ,tnirperp viXra " ,sevitcepsreP dna segnellaH ,sehcaorppA ,soiranecS :gnikaM .2503.13415 :viXra .gnikam-noisiced tnegA-itlum xelpmoc ot tnegA-elgnis morf
- [31] ni noituloseR tcilfnoC dna noitarepooC ,noitanidrooC` ,sgninneJ .N dna egdirdlooW .M noitanidrooc SAM no krow lanoitadnuoF ,noitamotuA fo koobdnaH ni " ,smetsyS tnegA-itluM 10.1007/978-1-4020-6268-1\_87 :IOD .7002 ,regnirpS ,smsinahcem
- [41] " ,sMLL fo yevruS A :smsinahceM noitaroballoC tnegA-itluM` ,la te ,lasnaB .G ,uW .Q ,epyt ,srotca yb snoitaroballoc gniziretcarahc yevrus evisneherpmoC ,5202 ,tnirperp viXra .2501.06322 :viXra .ygetarts dna ,erutcurts
- [51] viXra " ,noitartsehcrO gnivlovE aiv noitaroballoC tnegA-itluM` ,la te ,uiL .C ,gnahZ .W noitaroballoc tnegA-itlum desab-MLL rof mgidarap elyts-reetteppup sesoporP ,5202 ,tnirperp .2505.19591 :viXra .rotartsehcro deniart-LR htiw