File Breakdown: `src/agents/ivan.py`

File Location

`src/agents/ivan.py`

Overview

The `ivan.py` file implements the Ivan agent, which specializes in code generation, Python scripting, and image generation. Ivan extends the BaseAgent class to provide technical capabilities including generating Python scripts and creating visual content. It serves as the system's creative and technical assistant for visual and code-related tasks.

Key Responsibilities

- Generate Python scripts based on user requirements
- Create images based on textual descriptions
- Handle task execution both synchronously and asynchronously
- Collect missing parameters from users when needed
- Store code and image generation results in the knowledge base

Core Functionality

Class Definition

```python
class Ivan(BaseAgent):
    # Ivan agent extends BaseAgent with code and image generation capabilities
```

Task Handling

Synchronous task handling implementation:

```python
@log_function_call
def handle_task(self, task: Task):
    """
    Synchronous version of handle_task.

    Parameters:
        task (Task): The task to handle

    Returns:
        The result of executing the function or None
    """
    if task.function_name in self.function_map:
        func = self.function_map[task.function_name]
        missing = []
        for param in func.__code__.co_varnames[1:]:
            if param not in task.args:
                missing.append(param)
        if missing:
            new_args = self.ask_user_for_missing_args(missing)
            task.args.update(new_args)

        # Call the function with the args
        result = func(self.kb, **task.args)
        task.result = result
        return result
    else:
        print(f"Ivan doesn't recognize function {task.function_name}")
        task.result = None
        return None
```

Asynchronous task handling with special handling for image generation:

```
@log_function_call
async def handle_task_async(self, task: Task):
    """
    Asynchronous version of handle_task.
    Enhanced with better logging and categorized storage.
    """
    print(f"Ivan handling task asynchronously: {task.name}")

    # Log the start of task execution
    self.kb.log_interaction(f"Task: {task.name}", "Starting execution", agent="Ivan",
function=task.function_name)

    # Check for image generation requests
    if task.function_name == "generate_image":
        try:
            # Run the image generation function in a thread pool
            result = await asyncio.to_thread(self.generate_image, self.kb,
**task.args)

            # Store the result
            task.result = result

            # Store in knowledge base with categorization
            await self.kb.set_item_async("image_result", result,
category="image_generation")
            await self.kb.set_item_async("final_report", result)

            # Record in the session
            current_session = self.kb.get_item("current_session")
            if current_session:
                session_data = self.kb.get_item(f"session_{current_session}")
                if session_data:
                    # Create images_generated array if it doesn't exist
                    if "images_generated" not in session_data:
                        session_data["images_generated"] = []

                    # Add image info to the session
                    image_info = {
                        "timestamp": datetime.datetime.now().isoformat(),
                        "prompt": task.args.get("prompt", "unknown"),
                        "enhanced_prompt": self.kb.get_item("last_dalle_prompt")
                    }
                    session_data["images_generated"].append(image_info)
                    self.kb.set_item(f"session_{current_session}", session_data,
category="sessions")

            # Log successful completion
            self.kb.log_interaction(f"Task: {task.name}", "Image generated
successfully",
                                agent="Ivan", function="generate_image")

            return result
        except Exception as e:
            error_message = f"Error generating image: {str(e)}"
            print(error_message)

            # Log the error
            self.kb.log_interaction(f"Task: {task.name}", error_message, agent="Ivan",
function="generate_image")

            task.result = error_message
            return error_message

    # Regular task processing for non-image tasks...
```

Image Generation

A key feature of Ivan is the ability to generate images from text descriptions:

```
@log_function_call
def generate_image(self, kb, prompt):
    """
    Generate an actual image based on the provided prompt.
    Uses OpenAI's DALL-E API to generate the image and displays it.

    Parameters:
        kb (KnowledgeBase): The knowledge base
        prompt (str): The image description prompt

    Returns:
        str: A message about the generated image
    """
    import os
    import sys
    import tempfile
    import webbrowser
    from PIL import Image
    import io
    import base64
    import requests
    from datetime import datetime
```

```python
    print(f"Ivan generating real image for: '{prompt}'")

    # Better subject extraction with preservation of key terms
    subject = prompt.lower()

    # Remove only the command words but keep meaningful content
    command_words = ["create", "generate", "make", "draw", "show", "produce"]
    request_words = ["an", "a", "the", "some", "image", "picture", "photo",
"illustration", "of", "about", "showing", "depicting"]

    # First handle common phrases by replacing them with placeholder
    subject = subject.replace("solar pv", "SOLAR_PV_PANELS")
    subject = subject.replace("wind turbine", "WIND_TURBINES")
    subject = subject.replace("hydro power", "HYDROELECTRIC_DAM")

    # Remove command words
    for word in command_words:
        subject = subject.replace(word + " ", "")

    # Remove request words
    for word in request_words:
        subject = subject.replace(" " + word + " ", " ")

    # Clean up and restore placeholders
    subject = subject.strip()
    subject = subject.replace("SOLAR_PV_PANELS", "solar photovoltaic panels")
    subject = subject.replace("WIND_TURBINES", "wind turbines")
    subject = subject.replace("HYDROELECTRIC_DAM", "hydroelectric dam")

    # Generate subject-specific prompts for common energy topics
    if "solar" in subject or "pv" in subject:
        enhanced_prompt = "A photorealistic image of solar photovoltaic panels
installed on a rooftop, with blue silicon cells in aluminum frames capturing sunlight.
High-detail textures showing the glass surface and electrical connections."
    elif "wind" in subject:
        enhanced_prompt = "A photorealistic image of modern wind turbines on a green
field or offshore, with large white blades rotating in the wind against a blue sky.
Detailed view showing scale and mechanical components."
    elif "hydro" in subject or "dam" in subject:
        enhanced_prompt = "A photorealistic image of a hydroelectric dam with water
flowing through turbines, generating clean electricity. Detailed view showing the
massive concrete structure and power generation equipment."
    elif "energy" in subject or "power" in subject:
        enhanced_prompt = f"A photorealistic, high-quality image of {subject}, showing
detailed technological components and environmental context, with realistic lighting
and textures."
    else:
        # Generic prompt for other subjects
        enhanced_prompt = f"A photorealistic, detailed image of {subject} with
professional lighting, accurate proportions, and high-resolution details."

    # Log the enhanced prompt
    print(f"DALL-E PROMPT: '{enhanced_prompt}'")
    kb.set_item("last_dalle_prompt", enhanced_prompt)

    try:
        # 1. Create the image using OpenAI's DALL-E API
        from utils.get_api_keys import get_api_key
        api_key = get_api_key('openai')

        headers = {
            "Content-Type": "application/json",
            "Authorization": f"Bearer {api_key}"
        }

        payload = {
            "prompt": enhanced_prompt,
            "n": 1,
            "size": "1024x1024",
            "response_format": "url"  # Get URL rather than base64 for efficiency
        }

        print("Sending request to DALL-E API...")
        response = requests.post(
            "https://api.openai.com/v1/images/generations",
            headers=headers,
            json=payload
        )

        # Handle the API response, download and display the image...
    except Exception as e:
        # Fallback to ASCII art in case of error
        print(f"Error generating image: {str(e)}")

        # ASCII art implementation and error handling...
```

Key Features

1. **Image Generation**: Creates visual content using OpenAI's DALL-E API
2. **Prompt Enhancement**: Improves image generation prompts for better results
3. **Async Support**: Provides asynchronous task handling for non-blocking operations
4. **Fallback Mechanisms**: Includes ASCII art fallbacks for cases when image generation fails
5. **Parameter Collection**: Dynamically collects missing parameters from users
6. **Knowledge Base Integration**: Stores generated content in the knowledge base
7. **Session Tracking**: Maintains records of generated content in the session history

Integration

- Communicates with the knowledge base to store generation results
- Uses external APIs like DALL-E for image generation
- Coordinates with the task manager for task execution
- Interfaces with the file system to save generated images
- Uses the browser to display generated content

Workflow

1. Receives a task from Nova related to image or code generation
2. Checks for missing parameters and collects them if needed
3. For image generation:
   - Processes and enhances the prompt
   - Calls the DALL-E API
   - Saves and displays the resulting image
   - Provides a fallback (ASCII art) if generation fails
4. For code generation:
   - Generates the requested Python script
   - Stores it in the knowledge base
5. Records the results in the session history
6. Returns the formatted result to the task manager

Implementation Notes

- Uses thread pools to run synchronous functions in asynchronous contexts
- Implements detailed error handling with fallback mechanisms
- Provides specialized handling for different request types
- Includes domain-specific knowledge for energy-related image generation
- Employs techniques to improve DALL-E prompt quality