01 Jan 2011

A Semantic Agent Framework for Cyber-Physical Systems

Jing Lin

Sahra Sedigh
Missouri University of Science and Technology, sedighs@mst.edu

Ann K. Miller
Missouri University of Science and Technology

A Semantic Agent Framework for Cyber-Physical Systems

Chapter · March 2011

DOI: 10.1007/978-3-642-18308-9_9

3 authors, including:

Sahra Sedigh Sarvestani
Missouri University of Science and Technology

105 PUBLICATIONS **719** CITATIONS

SEE PROFILE

A Semantic Agent Framework for Cyber-Physical Systems

Jing Lin, Sahra Sedigh, and Ann Miller
Department of Electrical and Computer Engineering
Missouri University of Science and Technology
Rolla, MO, USA, 65409
Email: {jlpg2, sedighs, milleran}@mst.edu

Abstract – The development of accurate models for cyber-physical systems (CPSs) is hampered by the complexity of these systems, fundamental differences in the operation of cyber and physical components, and significant interdependencies among these components. Agent-based modeling shows promise in overcoming these challenges, due to the flexibility of software agents as autonomous and intelligent decision-making components. Semantic agent systems are even more capable, as the structure they provide facilitates the extraction of meaningful content from the data provided to the software agents. In this book chapter, we present a multi-agent model for a CPS, where the semantic capabilities are underpinned by sensor networks that provide information about the physical operation to the cyber infrastructure. As a specific example of the semantic interpretation of raw sensor data streams, we present a failure detection ontology for an intelligent water distribution network as a model CPS. The ontology represents physical entities in the CPS, as well as the information extraction, analysis and processing that takes place in relation to these entities. The chapter concludes with introduction of a semantic agent framework for CPS, and presentation of a sample implementation of the framework using C++.

Keywords – cyber-physical systems, agent-based modeling, semantic capabilities, fault detection, multi-agent system, sensor networks, intelligent water distribution.

1. Introduction

The synergy between agent-based modeling and semantic technologies holds promise for the resolution of challenges posed by a broad range of complex systems, in particular cyber-physical systems (CPSs), where embedded computing and communication capabilities are used to streamline and fortify the operation of a physical system [1]. In CPSs, sensors collect information about the physical operation of the system, and communicate this information in real-time to the computers and embedded systems used for intelligent control. These cyber components use computational intelligence to process the information and determine appropriate control settings for physical components of the system, such as devices used to control the flow of a physical commodity, e.g., water or electric power, on a line.

A fundamental challenge in research related to CPSs is accurate modeling and representation of these systems, especially as related to reliability. Simplistic models that assume components fail independently are rendered unusable for the majority of CPSs, due to significant interdependencies within the cyber and physical infrastructures, respectively, and across the cyber-physical boundary. In other words, modeling of any CPS is hampered by the need to model both the cyber (software, communication network, computing hardware) and the physical infrastructure (physical components and their interactions). Furthermore, the application of graph-theoretic models is complicated by heterogeneity in the notion of "flow" in CPSs. "Information" is the flow on the cyber infrastructure that provides communication and computing capabilities. The flow on the physical infrastructure is domain-specific, e.g., power for an electric power grid or vehicles for a ground transportation system. Both types of flow need to be represented accurately, such that effects of any event are reflected in either or both networks. Thirdly, existing explicit communication protocols used to impart information between the cyber and physical infrastructures do not fully capture the semantics of the interaction between the two. The vision of using distributed computing resources in the cyber networks to manage the distributed resources in the physical infrastructure further complicates modeling of CPSs.

Among existing techniques, agent-based modeling holds promise in surmounting the aforementioned challenge, due to its capability of encapsulating diverse attributes within one agent, as well as its emphasis on the interaction among autonomous, heterogeneous agents, which share a common goal achieved in a distributed fashion. Sensors are the key to this approach, as they provide situational awareness to the agents and enable them to function based on the semantics of their mission and the specifics of their environment. The research presented in this book chapter aims to accurately model a CPS as a multi-agent system, where each agent is an independent entity that manages resources within its local scope. In the proposed model, information from the sensor networks is dynamically integrated with semantic services to support real-time decision support in the information-rich environment of a CPS.

The CPS domain used as a case study for an application of this model is intelligent *water distribution networks* (WDNs). In a WDN, physical components, e.g., valves, pipes, and reservoirs, are coupled with the hardware and software that supports intelligent water allocation. Fig. 1.1 depicts a sample WDN.

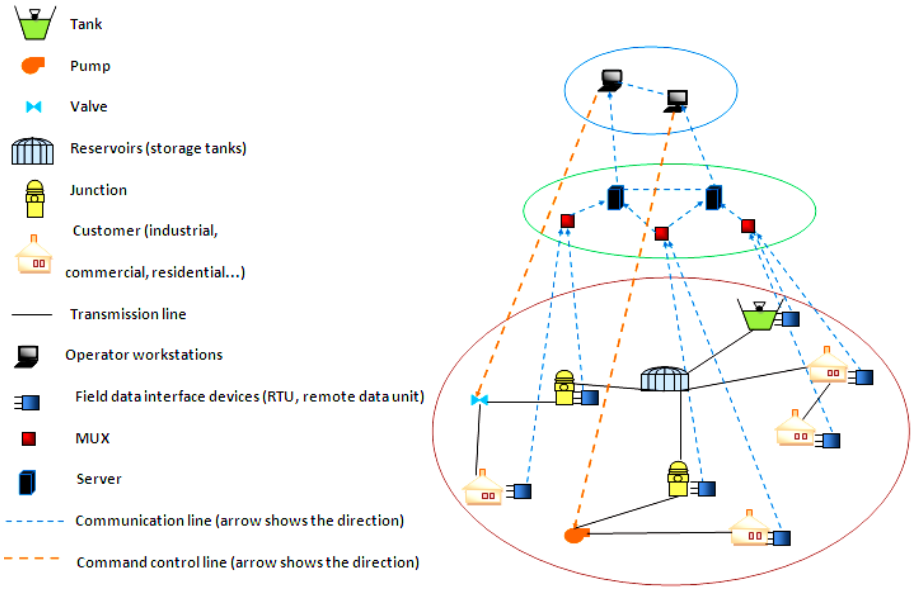

Fig. 1.1 Cyber and physical components of an intelligent WDN

The primary goal of WDNs is to provide a dependable source of potable water to the public. Information such as demand patterns, water quantity (flow and pressure head), and water quality (contaminants and minerals) is critical in achieving this goal, and beneficial in guiding maintenance efforts and identifying vulnerable areas requiring fortification and/or monitoring. Sensors dispersed in the physical infrastructure collect this information, which is summed by multiplexers and servers for hierarchical semantics interpretation. The processed and reasoned sensor data is then fed to distributed algorithms running on the cyber networks. These algorithms provide decision support to hardware controllers that are used to manage the allocation (quantity) and chemical composition (quality) of the water. The algorithms are implemented through software executing on multiple distributed computing devices. This software is represented by the agents in our model, each of which is capable of perceiving its environment, acting on that perception, communicating with other agents and exhibiting behavior that fits its goal.

This book chapter is an extension of our previous work, where we first articulated the use of semantic agents in modeling CPSs [2]. The extended content includes: a more detailed discussion on the agent-based modeling technique and its use in addressing design challenges in complex CPS, a more comprehensive presentation of our work on construction of semantic agent framework, and an introduction to data type processing.

The remainder of this book chapter is organized as follows. Section 2 presents an overview of related literature. In Section 3, we present tools and procedures

to construct an agent-based model, the method for defining an agent, and a UML multi-agent model that captures the static structure and dynamic behavior of a WDN. The semantic interpretation service is elaborated upon in Section 4, where the sensor information ontology and associated semantic service model are defined. In Section 4, we also propose a semantic agent framework for interpretation of the semantics of raw data streams, describe data type processing of the raw data stream, and provide an overview of implementing semantic interpretation capabilities through C++ on Matlab. Section 5 concludes the book chapter and describes future research directions.

2. Background Work

CPSs are an emerging research area, and the body of related literature is limited. A considerable fraction of related work examines critical infrastructure systems, which are prime examples of CPSs. Salient studies, e.g., [3, 4, 5, 6] are on interdependencies among different components of critical infrastructure systems, and [3], which provides a relatively comprehensive summary of modeling and simulation techniques for critical infrastructure systems. System complexity has been identified as the main challenge in characterizing interdependencies in CPSs [4]. Other challenges include the low probability of occurrence of critical events, differences in time scales and geographical locations, and the difficulty of gathering the accurate data needed for modeling. These challenges are clearly articulated in the literature, but solutions are very scarce.

The need to use agent-based modeling for distributed complex system has been investigated in [7]. The work in [8] adopts a distributed multi-agent architecture to analyze the observed information in real-time to adapt the multi-agent system to the evolution of its environment. To address the dependability issue in multi-agent system, [9] improves the capability of calculating how critical an agent is to the system through its interactions with other agents and provides a framework that uses this information to ensure availability and reliability. A multi-agent system (MAS) in [10] represents a powerful model to solve distributed computation problems. A particularly relevant study is presented in [11], where agent-based modeling is used to estimate residential water demand. An agent community is assigned to behave as water consumers, and econometric and social models are incorporated for estimating their water consumption. However, this study considers the WDN as a purely physical system with no cyber control.

As a formal specification language with precise semantics, UML 2.0 has been adopted to model multi-agent systems with precise semantics. A detailed demonstration of how UML 2.0 can be used for the specification of an agent-based system has been presented in [12]. UML 2.0 has been adopted during the analysis and design phase in [13], to model the physical and social contexts for embedded multi-agent systems. The specification of Action Semantics (AS)

in [14] shows how the applicability of AS to the UML meta-model paves the way for powerful meta-programming for model transformation.
Semantic agent technologies are typically closely associated with sensor networks, and several prototype systems or software architectures have been proposed based on the combination of the two. A prototype for battlefield information systems has been described in [15], where the stated goal is to dynamically integrate sensor networks with information fusion processes to support real-time sensing, interpretation, and decision-making in an information-rich tactical environment. In [16], an architecture and programming model for a semantic service-oriented sensor information platform has been presented. In contrast to [16] our work expands the semantic service model to a semantic agent framework, whereas [16] focuses on how to use the semantic model to query the system for high-level events without processing raw sensed signals. The use of autonomous semantic agents in developing new software architecture for distributed processing environments has been proposed in [17]. The discussion in [17] involves software architecture in general, and utilizes semantic web technologies; whereas our work is tailored to the specific requirements of CPSs. Due to the stringent security requirements of critical infrastructure and the vulnerabilities of web technologies, we do utilize them at this stage of our research.
The complexity of CPSs, as well as the necessity of capturing embedded computing and communication capabilities motivates the use of distributed agents and semantic services for representing the relationship between the cyber and physical infrastructures. In our work, the distributed semantic agent model augments the data acquisition of sensors in the CPS with ontological decision-making intelligence. The proposed model not only captures the complexity of the CPS in a clear and understandable way, but also takes accurate semantic interpretation into consideration. To our knowledge, our work is the first study to apply semantic agents to modeling of CPSs.

3. Agent-based Modeling technology

As a visual modeling language for representing object-oriented systems, UML is an intuitive choice for supporting agent-based modeling, in both the design and the communication phases. UML consists of several types of structured diagrams and graphical elements that are assembled to represent a model. The high level of abstraction is independent of the implementation of the model, especially when an object-oriented programming language is used.
Generally, by our definition, the agent is a piece of software code with intelligent decision-making functionality. An agent can be considered a self-directed object with the capability to autonomously choose actions based on its situation, and therefore the object-oriented paradigm is a useful basis for agent modeling. Object classes can be used as agent templates, and object methods

can represent agent behaviors. The data-driven, rather than process-driven perspective of object-oriented modeling also makes it well-suited to agent-based modeling.
The construction of an agent-based model can be broken down into the following steps, each of which is described in one of the subsections that follow.

1) Defining the agents in the context of the system; and identifying attributes of the agent; and other classes, along with their attributes.
2) Defining the environment where the agents reside, and the objects with which the agents interact.
3) Designing the methods by which agent attributes will be updated in response to agent-to-agent interactions or agent interactions with the environment.
4) Implementing the designed agent model in modeling software.

Our work specifically defines agent as software code and differentiates agents from the other devices, such as sensors and actuators. In contrast, architectures proposed in a number of other studies place the agents in the context of embedded devices. For example, in [18], the agent construction model is composed of components that are the basic building blocks for an agent; and the generic functionalities of these components are further divided into information collection (sensors), information storage (infostores), decision-making (controllers), and affecting change in the environment (actuators). In this book chapter, our focus is on the software aspects of agents, particularly on the implementation of semantic interpretation. Related work takes a more application-specific approach, e.g., the study in [19] discussed software aspects of information agents in a pervasive computing environment.

3.1 Definition of the agents

In our defined context of WDNs, the agent is the software code embedded in one or more computing devices on the cyberinfrastructure, with the goal of exerting control over components of the physical infrastructure. Other than this software, all other system components or subsystems are mechanical or hardware parts, including all mechanical water facilities, e.g., pumps, valves, reservoirs, water consumption junctions; communication links and sensors; or even more intelligent field-programmable gate array (FPGA) or programmable logic controllers (PLC) devices.

Regardless of their specific task, agents share the following characteristics.

1) An agent is an identifiable and discrete individual, as each segment of software code is located on distributed PCs to control a local water area. The subprogram code inherits the attributes and methods of the main program and develops its unique attributes or operations to manage water

resources within its scope. Therefore, it is constrained by rules governing its behavior, and in possession of decision-making capability.

2) An agent is situated in an environment where it interacts with other agents. In our model, each agent is in charge of its local scope, but they collectively interact for information sharing, data transmission and parallel computing.
3) An agent is goal-directed. Its major tasks include managing the raw data, using real-time data to quantify the overall reliability of the CPS, making a decision to take appropriate action if risk is anticipated in the near future, and sending control commands to actuators to meet the broader system objective or prevent potential damage. Approaches adopted for decision-making include game theory, which can be used to allocate water resources; the Leontief model [20], which can be applied to quantify the effect of a failure in one scope on operation of another scope; and Markovian models, which can estimate the likelihood of a transition from the current state to a given future state.
4) An agent is flexible, due to its nature as a segment of code. It can learn and adapt its behavior to the environment, based on new information, which includes data from sensors or from peer agents; and experience, such as data retrieved from a history database.
5) An agent is responsible for its intelligent semantic inference. After receiving raw data from the sensors, each agent should firstly check the integrity of the data, specifically, whether the data is deemed legitimate per scientific hydraulic relationships among its various physical parameters; and whether the data is reasonable, compared with history data and that of surrounding nodes. A large number of failures can be screened out through this procedure, and the redundancy of data can be greatly reduced through semantic aggregation.
6) An agent requires some form of memory, either on the computing device or in a separate database, to store the data of various water attributes for a period of time.

3.2 Construction of an agent-based model

In this section, we present an agent-based model for an intelligent WDN, as a case study of CPSs. We use the various types of UML diagram to gain insights to the system functionality, property and behavior of system components, software architecture, and the dynamics aspects of the complex system.

- **Use Case Diagram**

Creating a use case diagram is the first step for system analysis. A use case captures the interaction of a number of external actors with the system towards accomplishment of a goal. Fig. 3.1 shows the actor and the use cases involved

in the intelligent WDN. The use case diagram presented here can be generalized to other CPSs whose main goal is management of a physical commodity. Examples include power grids and intelligent transportation systems. As described in Section 1, the primary goal of WDNs is to provide a dependable source of potable water to the public. The specific role of the agents in the system is to intelligently guide water allocation, per the algorithm programmed in the agents.

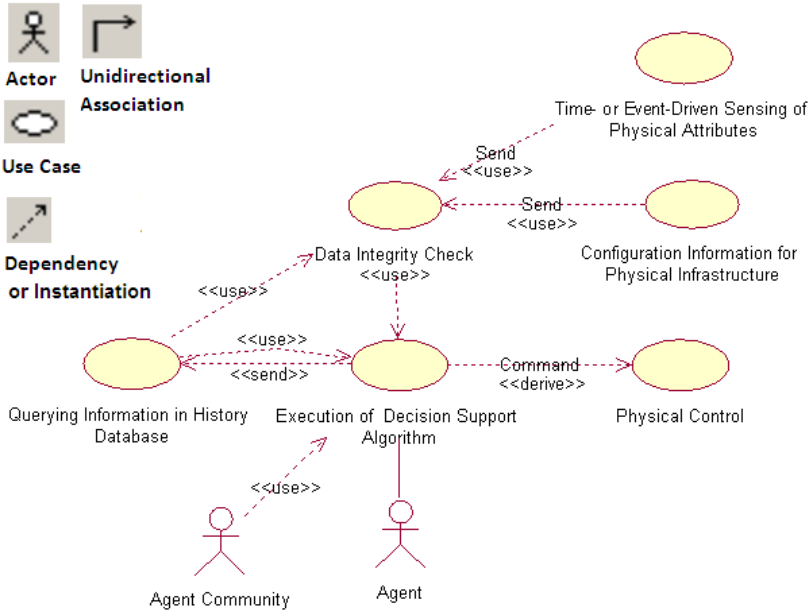

Fig. 3.1 Use case diagram of an intelligent WDN.

The CPS agent is the actor in the use case diagram, and associated with the decision support algorithm. For simplicity, only use cases associated with one agent are shown in Fig. 3.1; all other agents have similar use cases associated with them. As shown in the use case diagram, sensors collect information about the physical operation of the system on a time- or event-triggered basis. The collected information is aggregated by a multiplexer and sent to the *Data Integrity Check* for intelligent semantic inference. The *Data Integrity Check* use case uses three main data streams, specifically, raw data from the corresponding sensor, real-time data from nearby sensors for the same or related physical attributes, and the data from a history database. The second and third data streams mentioned are used for corroboration of the first, by checking for discrepancies in the values, whether in variation or in conformance to physical (hydraulic) laws that govern the physical operation of the WDN. If no data is available from nearby sensors, as would be the case if all nearby sensors are in sleep mode, the history database will serve as a source of data for corroboration. As indicated in Fig. 3.2, the values of physical attributes, such as water

quantity, of nearby nodes, should not be significantly different from each other for the same time period. For instance, adjacent water nodes should have similar water temperature and similar water pressure value. If significant discrepancy exists, the use case can conclude that the collected data may not be a legitimate group of data and should not be used for further information processing.

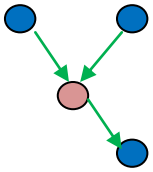

Fig. 3.2 Flow among nearby nodes

The decision support algorithm uses three data streams, one data stream from the *Data Integrity Check*, another from the history database, and a third data stream from other agents. The decision support algorithm is an advanced algorithm implemented through software code for intelligent management of physical commodities. The algorithm can make use of legitimate (corroborated) data whose integrity has been checked, and can also resort to history data for adjustment (rectification) of the calculated values in determining an appropriate strategy for resource allocation. Meanwhile, the local agent interacts and negotiates with other agents by sharing real-time information that provides global perspective of resources in the system, and adjusts its own strategy accordingly. For instance, one adjacent agent reports that pipe bursting have been detected and more water is needed from neighboring areas to guarantee regular water consumption before restoration. In this case, the well-being agents will adjust the strategy to maintain the local water consumption and support extra quantity of water to its neighbor. Various algorithms can be the candidate for the decision support algorithm, and the game theory holds the greatest promise.

- **Class Diagram**

Based on the use cases and interconnections defined in Fig. 3.1, Fig. 3.3 provides an overview of different classes in the intelligent WDN, along with the specified attributes and the corresponding methods for each class. Fig. 3.3 also depicts and how the classes interrelate. Other information provided in Fig. 3.3 includes the data types of the attributes and the main constraints used in the decision making algorithm. The attributes of the water facility classes have been chosen to be most representative of both static (elevation) and dynamic aspects (head loss) of water.

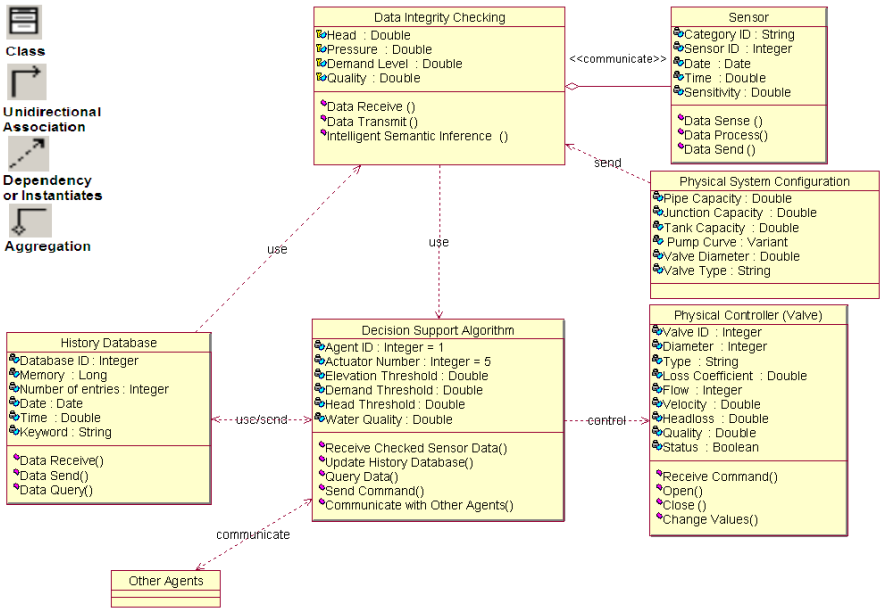

Fig. 3.3 Class diagram of an intelligent WDN.

The *Data Integrity Checking* class takes three data streams, from *Sensor*, *Physical System Configuration* and *History Database*, respectively. Data collected by the sensors is aggregated by the multiplexor (representing by the small diamond) and sent for data integrity checking. The *Physical System Configuration* block specifies the basic configuration and topology physical water infrastructure. This configuration data is sent to *Data Integrity Checking* to assist in evaluating physical constraints, e.g., judging whether a newly requested water value (such as quantity) will exceed the capacity of a pipe. History data can be queried by the *Data Integrity Checking* for comparing abnormal real-time data with historical values. Various types of semantic analysis are carried out through *Intelligent Semantic Inference*, including the aforementioned evaluation of physical constraints and corroboration with historical data or data from nearby nodes.

The purpose of this semantic inference is to screen out illegitimate or corrupted data (based on the preliminary judging criteria), to ensure that only legitimate data is sent to the decision making algorithm. A domain ontology for more advanced semantic interpretation and system failure detection based on semantic interpretation will be introduced in Section 4.

The agent has varied types of association with other classes: it receives the data after semantic processing, stores the data in the history database or queries data from the database to assist in decision making (bidirectional), negotiates resource allocation with other agents, and exerts control over actuators (valves and pumps).

- **Component Diagram**

In Fig. 3.4, the main program that implements water allocation executes on the cyberinfrastructure. The physical location of the main program is immaterial. The main program is directly dependent on the code specification, which is the head file of the agent class. It includes prototype information for the class function. The remainder of the script is the package body, which exhibits functionality similar to that of the main program and executes in distributed fashion within its autonomous management scope. If the script is written in C++, the package body is a .cpp file. An independent database is attached to each script, meaning that the script can only retrieve data from or store data to the database for management purposes within its own scope. All the data sent to the script for advanced semantic analysis or decision making has been checked its integrity, as described earlier in this chapter.

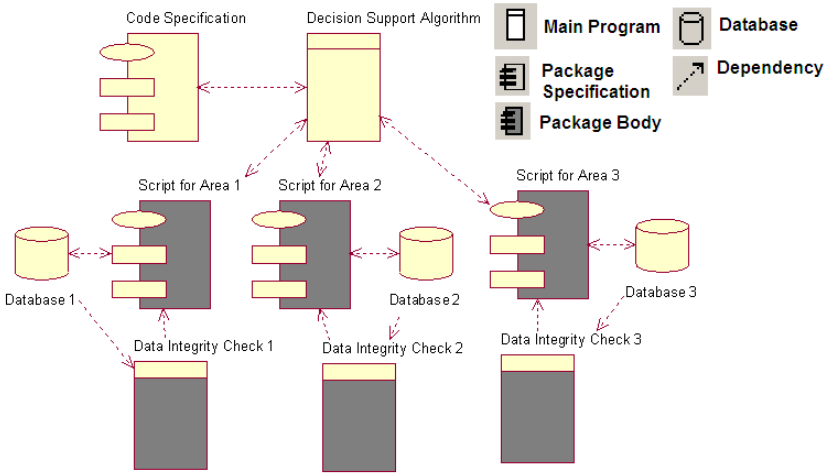

Fig. 3.4 Component diagram of an intelligent WDN.

- **State Transition Diagram**

Fig. 3.5 depicts the state transitions of data in one period, which is the time span from the point that data is collected at preset time (start state) until control has been exerted on the water consumption entity (end state). As agent-based modeling is a data-driven modeling method, it is vitally important to track each state transition of the data. The condition that can trigger entry to or exit from a particular state has been specified. The history state (encircled 'H') records the state of the system immediately before query of the history database. Once the agent has finished data retrieval, the state reverts back to the original state before data storage, and the agent begins processing based on the combination of retrieved historical data and the originally collected data. The flow of the decision making procedure, whose goal is to allocate water (quan-

tity), has been specified in the figure with two decision blocks (encircled diamonds).

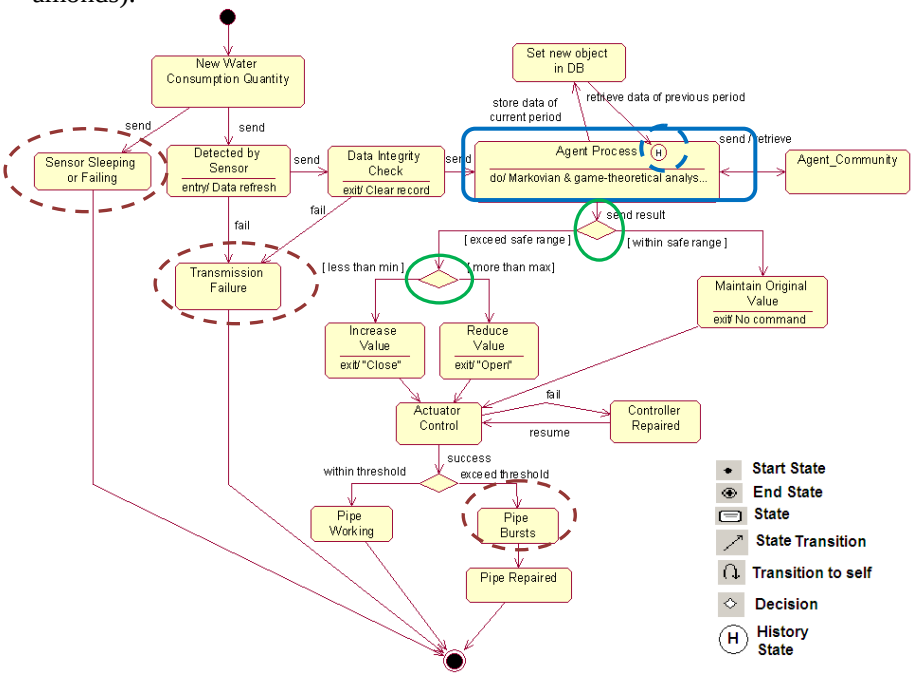

Fig. 3.5 State transition diagram of an intelligent WDN.

The *Agent Process* (in a solid rectangle) is the critical state within the context, as it provides a precise numeric value to guide the control over actuators. Markov and game-theoretic analysis are included in the state as instances. Since the state of the data within the system has already been identified through the agent-based model, we can use a vector to numerically represent these states. Failure is a state transition from functional to non-functional, such as the Sensor *Sleeping or Failing*, *Transmission Failure*, and *Pipe Bursts* states (all in dotted ellipses) shown in the figure. We can define the normal functioning state to be 1 and failure state to be 0. Therefore, vectors formed by 0 and 1 can precisely represent the state of the system. At this point, to identify the functional states, a Markov reliability model can be built to estimate the probability that the next state of the system is an operational state.

Game-theoretic analysis can be applied to calculate the equilibrium state of the water allocation among the agents. In the context of a city, the water quantity allocated to each sub-area can be determined, subject to the constraints of the physical facility. Base on the threshold values of the constraints, the optimal water allocation scheme can be obtained as well.

- **Activity Diagram**

In Fig. 3.6, which depicts the activity diagram for an intelligent WDN, three entities are involved, including the physical networks; agent 1, acting as the main agent; and agent 2 as the agent interacting with agent 1. The activity diagram reflects how an agent interacts with the environment, and how the values in the associated object change after date integrity checking and data processing.. For instance, the raw data is changed into semantically-processed data for control, and the requested water quantity of one agent may affect another agent's water consumption quantity.

- **Sequence Diagram**

Fig. 3.7 depicts the sequence of messages exchanged among different entities in the intelligent WDN. The message on the line shows the method adopted by the receiver (class defined in the class diagram) upon receiving the message. The figure shows the sequence of data received by the data integrity checking object and the decision support algorithm object of agent. For the former object, it directly receives and checks the raw data from the sensors (collected by multiplexor) and then if it needs to compare the real-time data with previous history data, it will receive data from its local database to make sure the result of checking is based on a reliable history record. The water consumer object and the adjacent agent object are eliminated after they send the return message, which means that no message from these two objects will be accepted outside of particular periods. The decision support algorithm of the agent first receives checked sensor data first, queries data from the history database, and finally communicates with the adjacent agent. Such a sequence is from the physical infrastructure to the cyber infrastructure (bottom-up). After the decision has been made, the calculated result will be sent to the community agent first, then a command will be sent to actuator to exert real-time control over the physical commodity, and finally the calculated data is recorded as history data in the database. Such a sequence is from the cyber network to the physical infrastructure (top-down), culminating in data recording.

A clear and correct sequence diagram of the agent-based WDN is the prerequisite for resolving challenges related to timing in CPS modeling. As the CPS is a two-layered system, the intelligent agents make decisions based on the collected data, but when the control command is sent back the actuator, the previous data for computation has already changed. Therefore, how to select an appropriate cycle period and how to process the changing data are open problems for further research.

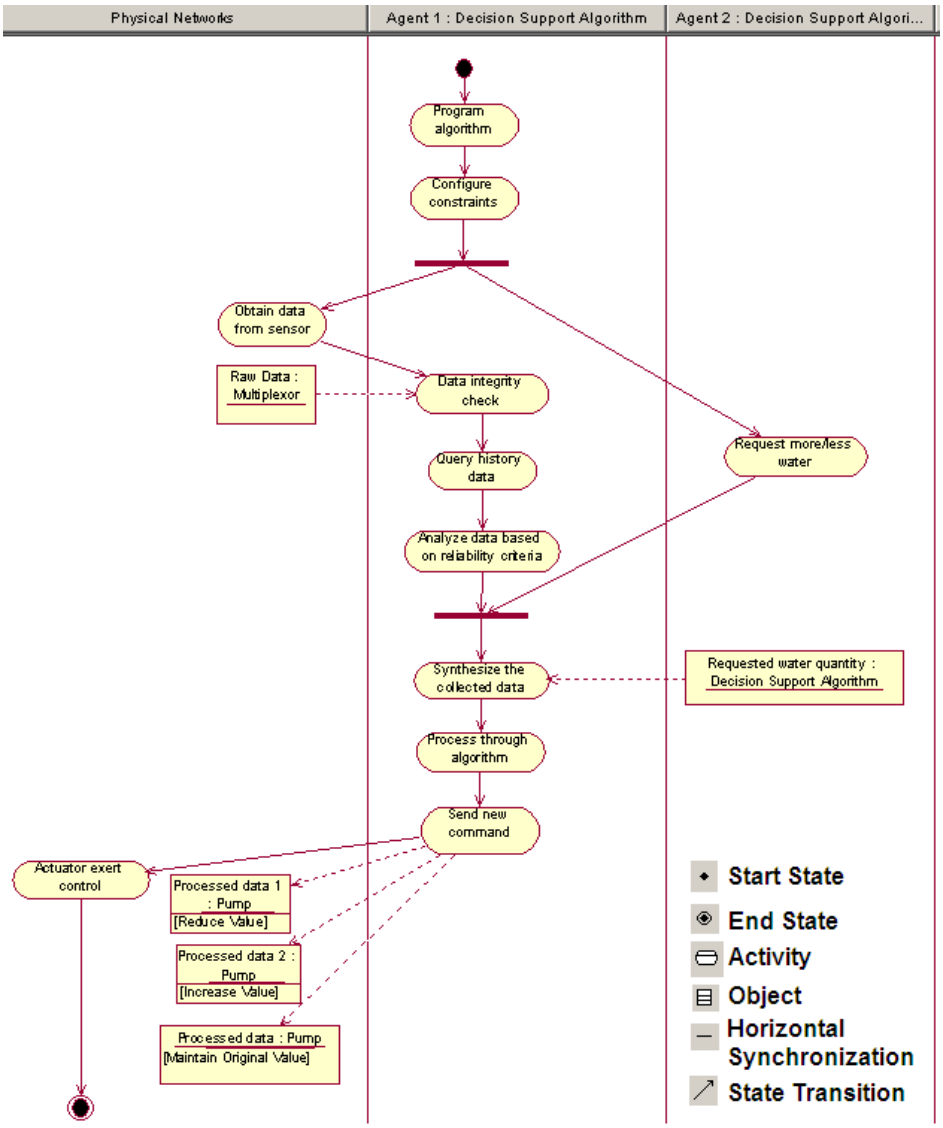

Fig. 3.6 Activity diagram of an intelligent WDN.

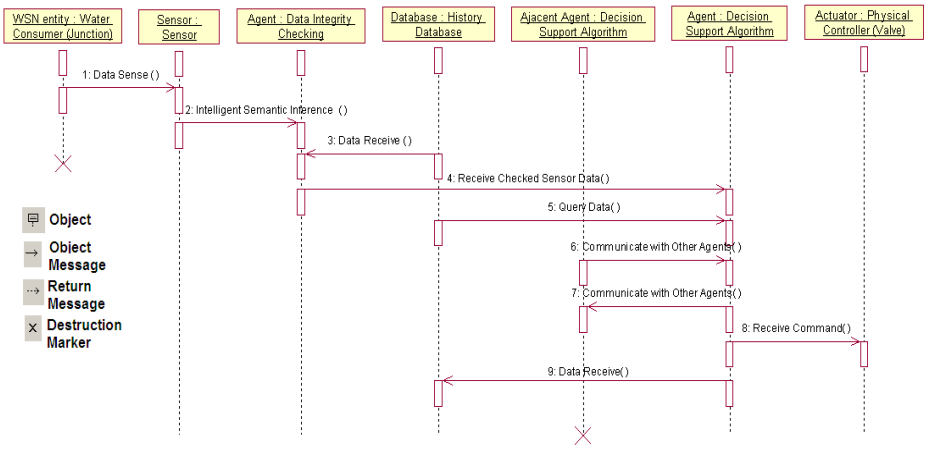

Fig. 3.7 Sequence diagram of an intelligent WDN.

4. Semantic Interpretation Services

4.1 Sensor information ontology

Semantic interpretation is carried out on semantic streams, each of which is defined in a domain-specific ontology associated with the agent. The specific domain in this book chapter is intelligent water distribution. Generally, an ontology is a description, e.g., a formal specification of a program, of the concepts and relationships that can exist for an agent or a community of agents. The notion of ontology utilized in this book chapter is a model that describes semantic relations among components of the physical and cyber infrastructures, respectively, as well as the interdependencies across the cyber-physical boundary. Each component in the ontology model is a unique class in terms of programming implementation, with properties and parameters described in the class definition. The relations define how classes can be related to one another. Semantic interpretation is implemented through distributed software with capabilities of extraction, analysis, and processing of the semantic stream. The definition of ontology for the WDN domain helps unify information presentation and permits software and information reuse, so as to reduce information redundancy during the process of semantic interpretation in the agents.

The use case diagram in Fig. 3.1 depicts intelligent control of the physical infrastructure by the cyberinfrastructure of an intelligent WDN. To achieve the goal of intelligent management and control, a number of tasks are involved to implement various functionalities (use cases), such as the pre-processing of the raw data from the sensors, coordinating the time sequence to query data from the history database and communicate with peer agents, converting the logical command to physical control over actuators, and so on. As ontology has advantage over other information representations in terms of capturing the

structure and meaning of information, we use ontology to represent the failure detection procedure, which is an important component of the intelligent information reasoning functionality in CPSs. Similar ontologies can be identified for other functionalities or use cases of a CPS.

Fig. 4.1 shows the information hierarchy for failure detection through the semantic interpretation process. In the UML class diagram, each block represents one type of semantic stream in the intelligent WDN. The attributes of each class have been omitted in the interest of figure clarity. Details of the attributes are presented in Fig. 4.6, which shows pseudo code for the semantic service.

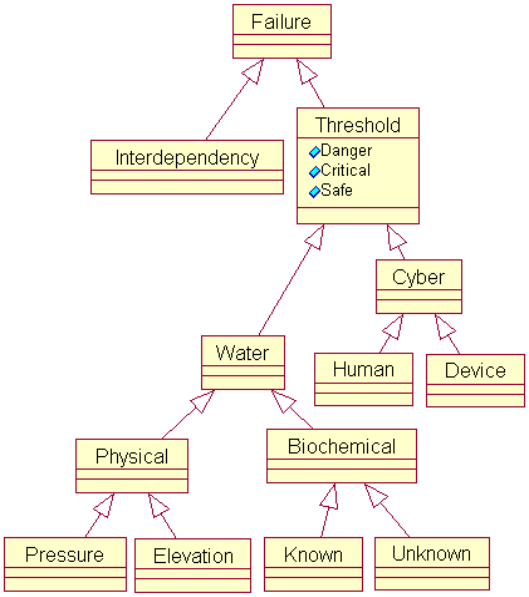

Fig. 4.1 UML representation of failure detection ontology in CPS for an intelligent WDN

Fig. 4.1 shows that a failure in the WDN can be detected by the agent in the event of device or information failure, the latter of which occurs when data falls outside a pre-defined safety range. Failures in the physical infrastructure of a WDN are of two main two types, physical failure due to excessive values of pressure and elevation, or biochemical failure due to excessive quantities of a biochemical substance or discovery of unknown biochemical materials. Cyber failures can be caused by human error or device malfunction. Each class identifies one type of semantic stream that can lead to failure in the CPS, and the ultimate determination of failure (or the overall interpretation) is carried out by the corresponding agent, which is in charge of all sensors deployed within its administrative scope.

The main reason that the attributes of the class have not been defined here is

for simplifying reasoning procedure on information. For instance, the danger threshold can be triggered by both excessive water quantity or cyber malfunction, but the excessive quantity of one single attribute of the water class is sufficient to diagnose the source of failure is from physical networks. Besides, the undefined attributes can help to reduce the semantic redundancy in terms of automatic semantic conversion, which not every property field of a class needs to be filled or met before performing detection. For example, to identify biochemical attack from the excessive biochemical quantity (such as excessive bacteria), the agent can just check if the detected biochemical element falls into the database of known elements. As long as one type of elements is unknown, even other co-existing elements fall into the knowledgeable scope, the agent can immediately determine that a failure can be caused by the unknown biochemical element.
The sensor information ontology captures the semantic entities (classes in the UML diagram) and the relations of events and objects, deriving a reasoning procedure beyond what sensors can directly provide through detection. The ontology proposed in Fig. 4.1 is specific to the WDN domain, but can be readily adapted to other CPSs, such as smart power grids.

4.2 Model for semantic services

Based on the sensor information ontology proposed, we can develop components that convert semantics between classes in the information processing hierarchy, by extracting new semantic information from existing data streams. In other words, the components encapsulate the semantic service into a ``black-box" containing the execution method, which takes as input information (defined as precondition [21]) corresponding to events detected by sensors and generates as output a number of meaningful new events (defined as postcondition [21]). The process is depicted in Fig. 4.2.

Fig. 4.2 Semantic service on the semantic stream

We propose a semantic service model that overlays the ontology defined in Fig. 4.1. The semantic service model allows the agents (users) to annotate semantics of data transmitted between the services of each entity on the ontology, and can check and automatically convert between data semantic whenever possible. As the services are event-driven, the events passing between services not only carry their value information, but also serve as triggers for service execution. The semantic services can be categorized into two types, i.e., a) the

service that supplements input events with additional semantic annotation, and b) the service that produces new semantic streams.

The first type of service can only identify additional properties carried by the input event. For example, a sensor has detected that the water pressure in a certain area has exceeded the safety threshold and reports this event to its semantic service component, which can be a superior sensor or multiplexer. The semantic service model associated with this component will add the geographical location as an additional identifier to distinguish this event from events reported from other areas. Such functionality is particularly useful for distributed control and management in the context of CPS, where a service may not correspond to a centralized component that physically exists on one device; it can be physically implemented on several distributed devices, but logically exists as a single service.

The second type of service automatically terminates the input semantic stream, and uses the generated output semantic stream as the new stream propagating on the ontology. The essence of this type of service is semantic transformation, where the input and output events are different classes in the ontology. One typical semantic transformation is generalization. For example, in Fig. 4.1, an excessive pressure quantity will be interpreted as physical failure due to an abnormal pressure value. Later on, the semantic stream of physical failure will be propagated to a higher level for ultimate decision making, instead of the semantic stream of abnormal pressure quantity, which no longer exists. This case will be illustrated by the code in Section 4.6. Another example can be the derivation of danger event by passing the threshold component, which abstracts the possible sources of dangers in terms of water failure and cyber failure. Such case falls into the category of generalization.

The benefits of proposing such a semantic service model on information ontology include the reduction of information redundancy, pre-processing and abstraction of data for the agent, and the facilitation of semantic query by a user. A user can issue a query that requests that a certain data stream with desired semantics be provided to a certain component device to diagnose whether failure exists on the queried level.

4.3 Semantic agent framework

Fig. 4.3 illustrates how the agents use the information detected by sensor networks and the interpreted semantics through components based on the defined ontology. Raw data is obtained from sensor networks, and since each agent is an independent entity in charge of a particular geographical area, the sensors located in distributed areas are managed by different agents (with possible overlap). For a semantic service component, the input semantic events are preconditions of the service. The postconditions, i.e., the processed output semantics, are provided to the agents for further computing.

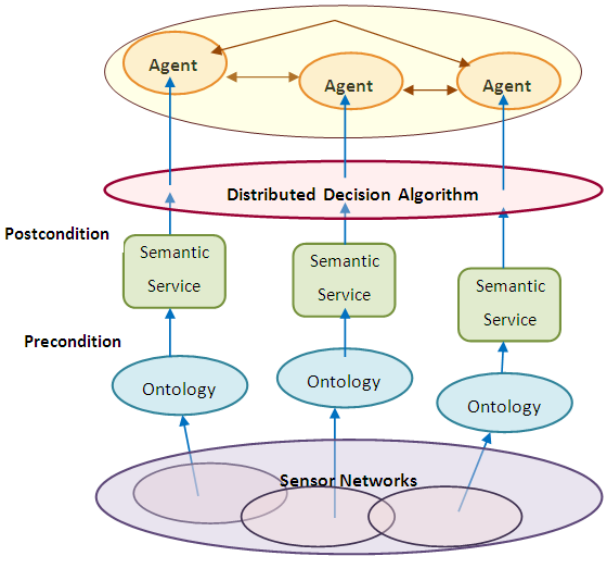

Fig. 4.3 Framework of semantic agents

The agents must host uncoordinated tasks, which are events triggered by different physical events and sensor data categorized in the ontology at unpredictable times. The data collected and process carried out strongly depends on the agents' surroundings. It is very likely that redundant information will exist among the concurrent tasks of different agents. For example, a pipe burst can impact several agents corresponding to nearby areas. Therefore, parts of the information for these tasks can be shared. To reduce the redundancy and use the computing resources more economically, we adopt a distributed decision algorithm, such as Maxflow, executing in parallel on multiple agents [22]. In the distributed decision algorithm, each agent uses only a portion of the computing resources to process the data within its own administrative scope, and the agents circulate the calculated results among each other to share information that may be helpful to the local decision making strategy.

A common application case is water allocation, and the algorithm we have adopted in the model is game theory. The details of our work have been presented in [23]. The sensors located in different areas collect water quantity information and send the data to the game theory algorithm. The water allocation strategy aims to achieve Nash equilibrium among the participating players - agents in the model. The equilibrium strategy seeks to achieve more efficient and fair water allocation, as compared with the low-level self-regulation that would occur if physical and hydraulic rules alone are left to govern the water flow. The proposed work can be further refined by introducing additional constraints, such as different pipeline thresholds and the maximum/minimum water quantity requirements of different water consumers.

However, some limitations exist in the model. The distributed algorithm is adopted by the agents due to limited computing resources, and the agent is responsible for the final decision used to exert control over the physical network. Several factors will impact this final decision. As shown in the use case diagram (Fig. 3.1), the execution of the decision support algorithm takes into consideration the data after integrity checking and comparison with historical values, and will also be affected by data from nearby agents. If the decision algorithm waits for data from all three sources to become available, the timeliness of the decision cannot be guaranteed, particularly if the speed of water flow is high and water attributes change rapidly. If the decision algorithm does not wait for data from all sources to be ready, its implementation may suffer the risk that the final decision does not meet the requirements or violates constraints, as it will already be outdated by the time it is made. Designating the agents as the decision making authority should take such factors into consideration.

Another shortcoming of the hierarchical organization is information asymmetry. This problem is best illustrated through an example scenario, where the program segment used to manage a certain area crashes. As the computing resources and the algorithm are distributed, the code running on other computers should remain operational and unaffected by the failure. However, water flow is a dynamic commodity, and the areas managed by different agents are interconnected. The failure of even a single agent could lead to missing data, which in turn can lead to flawed decision-making by other agents, and potentially a cascading failure of a significant portion of the system. Such a scenario serves as a cautionary tale of the vulnerabilities introduced by the use of cyber control.

The fault tolerance of the model is limited by several factors. A number of them has are apparent from the state transition diagram of Fig. 3.5. Data collection by the sensors may suffer the risk of sensor failure or sleep, and information loss can be caused by transmission failure. Lost, delayed or incomplete data can directly affect the functionality of higher-level components. Besides, the cyber components suffer risks from computer crash, disconnect of communication links, and internal design issues of the decision algorithm, such as interference among the agents. Increasing the robustness of the system by addressing these issues is an open research topic.

Before the input events are processed in the semantic service model, the event stream will undergo data type processing, including data type definition, data type checking, and data type conversion. A number of issues related to timing synchronization and sampling remain to be resolved for the data type processing as well.

4.4 Data Type Processing

The main purpose of data type processing is to reduce runtime redundancies, based on event semantics. The events in the event-driven model serve two roles: carrying values and triggering further services defined in the failure detection ontology. It is crucial for agents to identify the maximum sensing overlap and to reduce runtime redundancy, which is an intermediate information reuse and summarization problem. In light of the ontology defined in Fig. 4.1 and the semantic agent framework of Fig. 4.3, the intermediate data processing should carry out three functions: a) identification of overlapping information from multiple sensor nodes, including those that collect physical water data, others that monitor communication links, and yet others that supervise the cyber infrastructure; b) suppress parts of the data that are useless for failure detection, keeping only the critical information active and sending it to the higher-level entity; and c) sharing intermediate data with its peer entities. The second and third attributes can be realized in the service-oriented architecture proposed in Fig. 4.3, and will be elaborated upon in Section 4.4 and Section 4.5. To maximally identify redundant sensing is the task of data type processing.

Based on inspiration from [21] and [24], we define an event as a tuple with two elements: a *value* and a *tag*. 2123In contrast to [24], we define a triggering event as a signal consisting of its respective value and tag types. The composition of an event can be represented as Fig. 4.4.

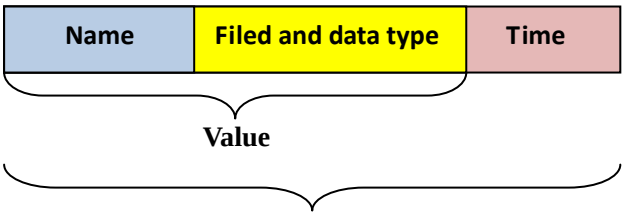

Fig. 4.4 Visual representation of an event

The value of an event is represented in the following form:

$$V= (name, \{n_1,p_1\},\{n_2,p_2,\ldots,(n_k,p_k)\}) , \qquad (4.1)$$

where *name* represents the name of the signal, and (n_i,p_i) represents the field and the data type. Such a representation of *value* can help parameterize the event and facilitate implementation of the service in the subsequent stage. For example, the *value* of an event that corresponds to measurement of physical attributes of water can be represented as:

water sensing = (water_sensor, {geoID, int}, {width, float}, {length, float}, {height, float}) (4.2)

Representation of events with a *value* also facilitates information aggregation. Based on record types in OCL programming languages [25], a pure specification language for expression, if two events have identical names, but the field type of one value is a subclass of the other, then the event value with a more generic field type can subsume the other event value to reduce data redundancy, while keeping the unique field type of the subclass. The following instance can be aggregated by (4.2).

water sensing = (water_sensor, {geoID, int}, {width, float}, {length, float}, {height, float}, {biochemical, float}) (4.3)

Aggregation of subclass data type is depicted in Fig. 4.5 (a), where the larger area denotes the common field type, and the smaller area denotes the unique field type of the subclass. After aggregation, the event record with the subclass field type does not exist. Information reuse, as an extension of information aggregation, keeps the common field type and the unique field type of the subclass separate, as shown as Fig. 4.5 (b).

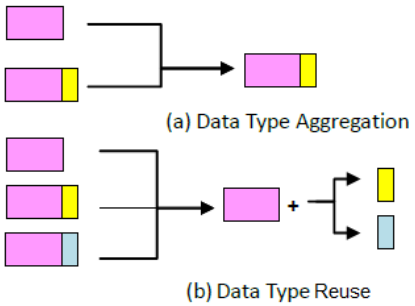

Fig. 4.5 Data type processing

Tags are utilized to represent timing and ordering relations among events. Sophisticated timing issues include the sampling frequency of sensors, difference and conversion between discrete time signals and continuous time signals, interpolation to merge different timing signals, and so on. More detailed information can be found in [24]. In this chapter, we simply use time, *T*, to represent the tag for each event.

Upon defining the value and tag, we can easily check and if necessary, convert the data type. In checking the data type, the focus is on checking the field type of the data, with the premise that the data has passed the integrity check. Data type checking can facilitate reuse of services from existing tasks, by comparing newly injected information about an event with existing information, in-

cluding matching the names of the event, checking for the existence of subset relations in the field type, and checking the data types in the field. The focus of data type conversion is mainly on reconciling the tag value of the triggering event with the tag value of the subsequently triggered event. As the tags dictate the timing and ordering relations among events, synchronization issues needs to be resolved, and the solution method depends on whether the triggered event is periodic or aperiodic. Interpolation is one solution method for periodic signals [24].

4.5 Implementation in C++

The choice of C++ for implementation of the semantic service was motivated by several factors. Firstly, a service component is an information-rich component that needs to define the semantic service for execution, extract information from the input, and produce new semantics at the output. Such logical analysis is best implemented through a high-level programming language such as C++ or JAVA, rather than a computing tool. As the modeling approach adopts UML, it is also natural to use the Object-Constrain Language [26] to specify the pre- and post- conditions and the actions in C++ or JAVA. Secondly, a class in C++ is a good fit for our definition of the service component; the declaration of service properties and the execution method of the service can be encapsulated into one class. Thirdly, Matlab2008b can integrate C++ and support parallel computing, and the integration of the semantic service and computation of the algorithm in Matlab will make simulation of the CPS more compact and faster.

To implement the service in C++, the properties of the service are parameterized, and the execution method of the service becomes the corresponding method in the service class. To illustrate the method, we choose the branch of *Pressure* to *Failure* in Fig. 4.1 as an example. Sensors are treated as services with only output semantics, which are parameterized into data that can be used by superior service components. Each component has been specified with a service name and a parameter associated with the service. Each service takes the outputs of an inferior component as the input to its execution method, and inherits the parameters to ensure that attributes of a potential failure source (such as pressure, failure time, or location) are not lost during information propagation on the ontology. The pseudo-script for C++ implementation is shown in Fig. 4. 6.

5. Conclusions

CPSs are the topic of emerging research, but existing tools and techniques for modeling them are still limited. A number of related challenges were discussed in this book chapter, with focus on the importance of capturing interdependencies and flow heterogeneity, and streamlining semantic interpretation between the cyber and physical infrastructures. The use of agent-based modeling was

proposed, and related methods and tools were introduced. An intelligent WDN was presented as a case study for demonstrating the ability of the technique to capture various facets of the operation of a CPS. A semantic service model based on the definition of ontology was presented, with the goal of reducing information redundancy and simplifying the data interpretation procedure of the agents. The data processing carried out for parameterizing and aggregating the raw data streams was described, as was the implementation of the semantic service model in C++. The proposed model reflects the semantics of intelligent water distribution, but can be modified for use in other CPS domains.
The modeling work presented in this book chapter is a preliminary step that will facilitate the broader goal of modeling CPSs. Future extensions to this work will incorporate sophisticated decision support algorithms, e.g., game theory, for the agents. The semantic service model implemented through C++ will be integrated with Matlab to facilitate the complex computation required. Provision of the semantic service in C++ to the decision support algorithm in Matlab will create an advanced simulation environment for CPSs, which can be invaluable to gaining a more profound understanding of the operation of CPSs.

```
Sensor{
water_sensor, geoID,[width,length,height], /* properties of sensors:
        water detection, geographical ID, location*/
Outputs(pressure, elevation, biochemical, location, T1);  /*the para-
                     meters can be detected by sensor at time T1* /
}

Pressure component:
Pressure_Service{
service(pressure),   /*service indicates execution method and the pa-
                              rameter is pressure*/
Inputs (sensor(water_sensor, geoID, [width,length,height], T1);
If (pressure > normal range)
Outputs (pressure_normal (false), detected (pressure,geoID, T2));
/*add the judgment result and time T2 when make judgment*/
}

Physical component:
Physical_Service{
service(physical_failure),
Inputs(pressure_service(pressure_normal(false), detected (pres-
sure,geoID,T2)));
If ((elevation < normal range) && (pressure_normal = false))
             /*guarantees pressure is the unique reason*/
Outputs(physical_normal(false), de-
tected(physical_failure,pressure,geoID,T2)); /*inherit inferior
attribute*/
}

Water component:
Water_Service{
service(water_failure),
Inputs(physical_service(normal(false), (physi-
```

```
cal_failure,pressure,geoID,T2)));
If ((biochemical_normal = true) && (physical_normal = false)) /*same
as above*/
Outputs(water_normal(false), detected(water_failure,physical
_failure,pressure,geoID,T2));
}

Threshold component:
Threshold_Service{
service(failure),
Inputs(physical_service(normal(false), de-
tected(physical_failure,pressure,geoID,T2)));
Switch(detected(pressure))
{
Case (within range for safe): service terminates;
Case (within range for critical): send (pressure,geoID,T2) to data-
base;
Case (within range for safe): output system failure alert;
Default: service terminates;
}
Outputs(system_failure_alert, detected( water_failure,physical
_failure,pressure,geoID,T2));
}
```

Fig. 4.6 Pseudocode for semantic service

References

[1] Lee E, "Cyber physical systems: Design challenges," in Proc. of the 11th IEEE International Symposium on Object Oriented Real-Time Distributed Computing, May 2008, pp. 363–369.

[2] Lin J, Sedigh S, and Miller A. "Modeling cyber-physical systems with semantic agents," the 5th IEEE Workshop on Engineering Semantic Agent Systems in conjunction with Proc. of the 34th IEEE International Computer Software and Applications Conference, Seoul, South Korea, July 2010.

[3] Rinaldi S M, "Modeling and simulating critical infrastructures and their interdependencies," in Proc. of the 37th Hawaii International Conference on System Sciences, 2004.

[4] Pederson P, "Critical infrastructure interdependency modeling: The survey of U.S. and international research," August 2006.

[5] Svendsen N K and Wolthusen S D, "Analysis and statistical properties of critical infrastructure interdependency multiflow models," in Proc. of the IEEE Information Assurance and Security Workshop, June 2007, pp. 247–254.

[6] Lin J, Sedigh S, and Miller A, "Towards integrated simulation of cyber physical systems: A case study on intelligent water distribution," in the 8th International Conference on Pervasive Intelligence and Computing, 2009.

[7] Macal M C and North J M, "Tutorial on agent-based modeling and simulation Part 2: How to model with agents," in Proc. of the 38th Winter Simulation Conference, 2006, pp. 73–83.

[8] Guessoum Z, Faci N, and Briot P J, "Adaptive replication of large scale multi-agent systems - towards a fault-tolerant multi-agent platform," in Proc. of the 4th International Workshop on Software Engineering for Large-Scale Multi-Agent Systems. ACM, 2005.

[9] C Gatti de M, Lucena de C, and Briot J, "On fault tolerance in law governed multi-agent systems," in Proc. of the 5th International Workshop on Software Engineering for Large-Scale Multi-Agent Systems . ACM, 2006.

[10] Poslad S, "Specifying protocols for multi-agent systems interaction," ACM Transac-

tions on Autonomous and Adaptive Systems, vol. 2, no. 4, November 2007.
[11] Athanasiadis I N, Mentes A K et al., "A hybrid agent based model for estimating residential water demand," Simulation, vol. 81, no. 3, March 2005.
[12] Bauer B and Odell J, "UML 2.0 and agents: How to build agent based systems with the new UML standard," Engineering Applications of Artificial Intelligence, vol. 18, no. 2, 2005.
[13] Klein F and Giese H, "Analysis and design of physical and social contexts in multi agent systems using UML," in Proc. of the 4th International Workshop on Software Engineering for Large-Scale Multi-Agent Systems. ACM, 2005.
[14] Sunye G, Le Guennec A and Jezequel J, "Using UML action semantics for model execution and transformation," Information Systems, vol. 27, 2002, pp. 445-457.
[15] Jiang G, Chung W, and Cybenko G, "Semantic agent technologies for tactical sensor networks," in Proceedings of the SPIE, 2003, pp. 311–320.
[16] Liu J and Zhao F, "Towards semantic services for sensor-rich information systems," in 2nd International Conference on Broadband Networks, 2005, pp. 44–51.
[17] Elci A and Rahnama B, "Consideration on a new software architecture for distributed environments using autonomous semantic agents," in Proc. of the 29th Annual International Computer Software and Applications Conference, 2005.
[18] Finin T, Joshi A et al., "Information Agents for Mobile and Embedded Devices," Lecture Notes in Computer Science, 2001, Volume 2182/2001, 264-286.
[19] Ashri R and Luck M, "An Agent Construction Model for Ubiquitous Computing Devices," in Proc. of the Fifth International Workshop on Agent-Oriented Software Engineering, 2004.
[20] Haimes Y Y and Jiang P, "Leontief-based model of risk in complex interconnected infrastructures," Journal of Infrastructure Systems, vol.7, No.1, March 2001, pp. 1-12.
[21] Lee E A and Vincentelli A S. "A framework for comparing models of computation". IEEE Transactions on CAD, 17(12):1217–1229, Dec. 1998.
[22] Armbruster A, Gosnell M et al., "Power Transmission Control Using Distributed Max-Flow," Proc. of the 29th International Computers, Software, and Applications Conference, 2005.
[23] Lin J, Sedigh S, and Miller A. "A Game-Theoretic Approach to Decision Support for Intelligent Water Distribution," Hawaii International Conference on System Sciences, January 2011.
[24] Liu J, Cheong E and Zhao F, "Semantics-based optimization across uncoordinated tasks in networked embedded systems," The International Conference on Embedded Software, September, 2005.
[25] Mitchell C J (1996) Foundations for Programming Languages. MIT Press.
[26] Object Constraint Language Specification Version 2.0, "OCL," http://www.omg.org/technology/documents/formal/ocl.htm. Accessed 20 February 2010.