A hybrid framework for the specification of automated material handling systems

Henry Y.K. Lau* and **Ying Zhao**†

Department of Industrial and Manufacturing Systems Engineering
The University of Hong Kong
Pokfulam Road, Hong Kong, PRC.

*e-mail: hyklau@hkucc.hku.hk
†e-mail: zhao-ying@hkusua.hku.hk

Abstract

This paper presents a hybrid framework that specifies and characterizes the capabilities of generic components in an automated material handling system (AMHS). The framework also provides rules and mechanism for binding these capabilities together so as to facilitate the process of task planning for AMHSs. As a hybrid framework, the formal mathematics of Communicating Sequential Process (CSP) is tightly integrated to the Unified Modeling Language (UML) to provide three important entities, namely, the object structure diagram, object communication diagram and CSP-based statechart to extend the capability of a UML model in specifying the key properties of AMHSs including synchronization, parallelism and communication. The results will bring us a step closer to the generation of a fully automated task-planning executive for AMHSs.

1 Introduction

An ultimate automated material handling system should compose of cooperating machines that communicate with one another to achieve a task autonomously. Such an automated material handling system (AMHS) is able to accept instructions, generate a feasible schedule, plan actions, and then executes the tasks using the functionality offered by the equipments such as conveyors, autonomous guided vehicles (AGVs) and shuttles. The essential knowledge that an AMHS control system requires includes the characteristics, constraints and capabilities of its components so that appropriate co-operation schedule can be determined. In this respect, a complete AMHS control system must be able to describe the characteristics of each AMHS components, and to capture key properties including synchronization, parallelism and communication during task planning. To develop such an AMHS control system, an appropriate specification and modeling framework is essential.

Object-oriented modeling techniques [1, 2], such as the Unified Modeling Language (UML), describe system in terms of self-contained entities associated with both the structural and the behavioral characteristics. They have been widely used in system design and analysis due to their characteristics including expressive modeling power, extensibility, reusability, and implementation independence, etc. While the advantages of using an object-oriented approach to describe and model a system are numerous, current object-oriented models have limitations in capturing the concurrent and dynamic semantics of automated systems. In particular, most object-oriented techniques do not have a formal means to precisely capture the key properties including synchronization, parallelism and communication during the specification and analysis of AMHSs. Communicating Sequential Process (CSP) [3, 4], on the other hand, is a formal method for describing concurrent systems with components interacting with one another. It has a well-defined behavioral semantics for describing synchronism and parallelism of concurrent systems. To combine the capacities offered by these two techniques in the system development process, a hybrid framework that integrates CSP and UML for characterizing the capabilities of generic material handling equipments, as well as providing rules and mechanism for binding these capabilities so as to facilitate the process of task planning and control system design is presented in this paper.

The paper is structured as follows: a brief description of the semantics and notations of CSP is given in the next section that is followed by the presentation of the proposed hybrid framework. Detailed discussion of the components of the framework is then followed and an application of the framework in specifying a typical material handling system is demonstrated through a case study.

2 Communicating Sequential Process (CSP)

The mathematics of CSP was introduced by Hoare [3] as a formal algebra for specifying concurrent systems. It defines a set of operators and axioms for describing discrete event processes, concurrency, non-determinism, and communication. A specification in CSP consists of $n \geq 1$ communicating processes; this is normally represented using the parallel composition operator ($\|$), which is associative: $P=\{P1\|P2\|.....\|Pn\}$. The parallel operators seen so far (||) has the property that all partners allowed to make a particular communication must synchronize for the event to occur. The opposite is true of parallel composition by interleaving, written as P|||Q. Here, the processes run completely independently to each other.

In CSP, processes communicate synchronously by sending and receiving messages through channels: the sending and receiving actions (or events) are specified using the input (?) and output (!) notations. Specifically, *P ? x* describes the action of receiving a value of *x* on a channel *P*, whereas, *P ! x* represents the action of sending a value of *x* through a channel *P*. Synchronization is accomplished by using the same channel for input and output actions in the two communicating processes. In addition, choice operators for selecting processes are defined for representing: (a) an external choices (□) where the selection is made by the environment, and (b) an internal choice (⊓) where such selection is made by the process itself. Specifically, there are two special CSP processes, namely, *STOP* and *SKIP*, which represent deadlock and successful termination of process execution respectively that are particularly important to the proposed framework that is introduced in this paper. Full descriptions of CSP notations and axioms can be referred in [3].

A number of studies of using CSP for system specification and verification have been carried out [5-7]. These include studies of the elevator control system [8], traffic control systems [9], and the computer network protocols [10]. Some attempts have been made to integrate the formalism of CSP and timed-CSP into object-oriented models for the design of real-time systems [11], and the specification of concurrent systems whose components have well-defined and modular behaviors [12]. These studies have established the practicality of CSP, both on its own and as an integrated method with an object-oriented model.

3 A hybrid framework for task planning

Tasks within an AMHS often involve multiple actions that are performed by interacting machines. These actions may involve transportation, storage, retrieval, sorting, packaging, checking of materials. One of the very important activities an AMHS control system performs is task planning in which two major issues are considered: formal decomposition of these tasks into sub-tasks and the allocation of these tasks to appropriate machine modules. An alternative perspective to task planning in an AMHS is to consider the capability of each individual AMHS components, groups of components, and the entire system in handling specific tasks. By mapping the desired task specification with the capabilities of AMHS components that compose the overall system, possible task plans can be generated.

As such the evaluation of the capabilities of AMHS is one of the key facets in the context of the proposed framework. In fact, the evaluation of the capabilities of AMHS components depends largely on the ability to characterize formally the properties of individual AMHS components. The proposed framework for capability modeling and task specification in the context of automatically plan tasks is illustrated in Figure 1. The framework consists of four main components with reference to the particular AMHS and the required material handling tasks. The task plans for the particular AMHS are being the outputs.

The central part of this framework is the hybrid formal object-oriented model, which has the ability to specify formally the essential behaviors and properties of the AMHS components and their interactions. Another important component is the capability model, which defines the semantics for specifying the capabilities of AMHS and provides rules for combing these component specifications based on the formal modeling approach of CSP. A repository of capability also forms part of the framework that is constructed by mapping the generic capability models to formal task primitives. This repository acts as a library that is used by the computer-aided tools for task planning. The computer-aided tools are also developed for building and evaluating of the capabilities of the AMHS.

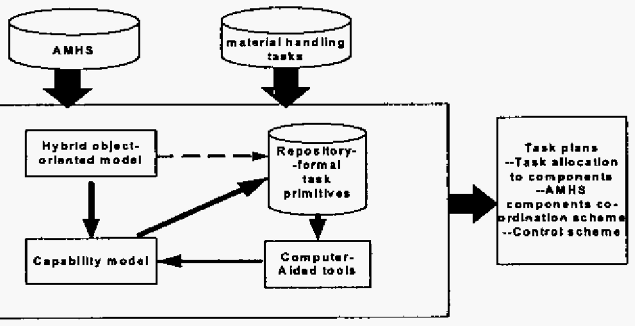
AMHS
material handling tasks
Hybrid object-oriented model
Repository--formal task primitives
Capability model
Computer-Aided tools
Task plans
--Task allocation to components
--AMHS components co-ordination scheme
--Control scheme

Figure 1: The capability modeling and task specification framework in the context of task automated planning

3.1 Formal hybrid object-oriented model

The proposed hybrid model is based on the notations of UML for object description, with the introduction of the concurrent process model of CSP for specifying the relationship between each object. Such integration extends the object model with the ability to specify the concurrency and communication aspects of concurrent processes. The formal hybrid object-oriented model consists of three main components, namely, the object structure diagram, the object communication diagram and the CSP-based statechart.

3.1.1 Object structure diagram: The conventional role of UML object diagram provides a structural architecture for objects and models the static associations between them. Based on the object diagram, CSP notations are introduced to formally describe the external dynamics of active objects in terms of external communication and parallelism. Such a representation is called the object structure diagram.

Specifically, the association in UML that is used to connect two objects together is modeled as an interface between two processes, precisely notated with appropriate CSP notations. Using the CSP notations for parallel composition (|| and |||), the object structure diagram describes not only the static but also the dynamic relationships between objects formally through the parallel (||) and the interleave relationships (|||). The association label is used to describe the set of external communicating events that occur at the interfaces of both processes. According to the formalism of CSP, both of these processes have control over the occurrence of these events. Figure 2 shows an example of object structure diagram. In this diagram, the processes of the *AGV* and *Storage* are operating in parallel and controlling the events of *load_part* and *store_good.*

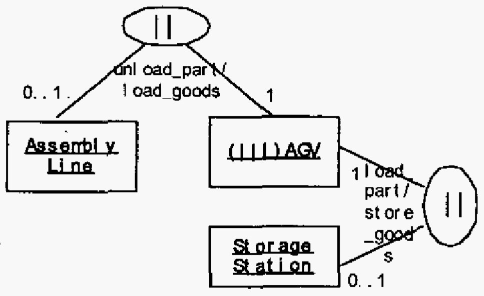

Figure 2: An object structure diagram

3.1.2 Object communication diagram: In order to capture internal communication, which is a main phenomenon of concurrent systems whose components interact with each other via information passing, an object communication diagram is developed.

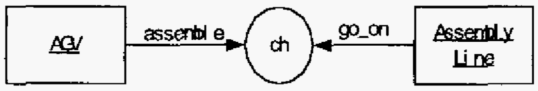

Figure 3: Object communication diagram

CSP provides a formal means for describing and reasoning about complex internal communication patterns, as it encapsulates the fundamental principles of communication in a simple and elegant manner. CSP exclusively uses channels to realize internal communication between processes. Channels are one-way, un-buffered, and fully synchronized artifacts for information exchange. Consequentially, the association in a UML object diagram is modeled as a CSP channel that serves as an interface between two processes, annotated with the name of the channel. The arrows between a channel and objects correspond to the flow of information. In Figure 3, *ch* represents the communication channel that enables signals to flow between the *AGV* and *Assembly Line*, and the annotations of these arrows specify the signals and messages. In this example, the *AGV* sends a signal of *assemble* to the *Assembly Line* and the *Assembly Line* sends the signal of *go_on* to the *AGV.*

3.1.3 CSP-based statecharts: To precisely capture the dynamic behaviors of objects, a CSP-based statechart is developed by extending the notations of the UML statechart.

CSP choice operators (,⊓) are incorporated to the UML statecharts to formalize the two types of choices namely, external choice and internal choice, that is, a choice made by the environment and by the process itself respectively. There are more than one outgoing transitions with a state involving a choice, and every transition out of this state involves a branch of activity. This notion is illustrated by Figures 4a and 4b. At state *P*, if event *a* occurs, the system transits to the state *P1*, otherwise, the system transits to state of *P2*. A start state signifies the creation of an object that marks the beginning of the object lifetime, whereas an end state signifies the termination of the object lifetime, which may takes two different types: either successful or unsuccessful termination that corresponding to a deadlock or liveness condition. The processes of SKIP and STOP are also added to the termination states to clear define these termination conditions. These are illustrated in Figure 4c. In addition, the CSP internal communicating events, such as *e?v*, and *e!v* are added to the state transition to provide a precise specification for the specific events that cause the state transition (Figure 4d).

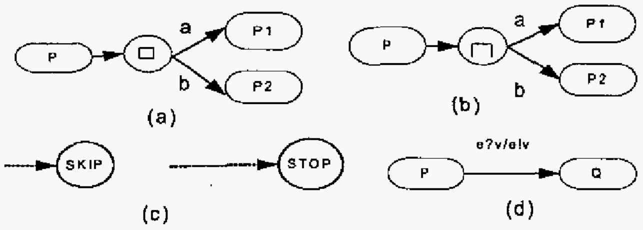

Figure 4: CSP-based statecharts

3.2 Capability models for task planning

In addition to the diagrammatic representation of the hybrid framework the semantics and notations for specifying the capabilities of an AMHS is given in this section.

Definition 1 – An Event

An Event is a discrete, observable and quantifiable entity that occurred instantaneously in time. There are two types of events, internal event and external event. The former cannot trigger the explicit state transition, while the state transition triggered by the later can be observed from outside the system.

Definition 2 – The Capability Set

The actions, that an object is capable of performing, are defined as the capability set. In general, the capability set of an object is composed of a set of finite or infinite discrete events, which happen sequentially or concurrently.

Definition 3 – A Capability Model

Under the hybrid framework, capability model is composed of a set of capability sets based on either external choice (□) or internal choice (Π), which is formally described by:

$$A \,\square\, / \,\Pi B \,\square\, / \,\Pi \ldots$$

Where A, B are capability sets

When events are triggered, an object changes its states; therefore a CSP-based statechart specifies the dynamics of individual object and these triggered events are mapped to the capabilities of the particular object. To illustrate the constructions of capability models, taking reference from Figure 4, the components shown have the following capability models:

$$a:\ \{a\} \,\square\, \{b\}; \quad b:\ \{a\} \sqcap \{b\}; \quad d:\ \{e?v/e!v\}.$$

Definition 4 – Combining Capability Models

Given the capability models of individual object, the capability of the overall system can be obtained by combing its component object diagrams based on the following rules, which are derived from the formal modeling approach of CSP.

Rule 1:

if $P = P1 \| P2$ *or* $P = P1 ||| P2$

the capability model of P1: $A_1 \,\square/\Pi\, A_2 \,\square/\Pi \ldots$

the capability model of P2: $B_1 \,\square/\Pi\, B_2 \,\square/\Pi \ldots$

then the capability model of P:

$$A_1 \cup A_2 \ldots \cup B_1 \cup B_2 \cup \ldots$$

where A_1, A_2, B_1, B_2 *are capability sets*

The capability model of the whole system should contain only those events through which the system interacts with its environment. The internal events take no parts at the system interface and are abstracted from the system *specification with the CSP hiding rule:*

Rule 2:

if $P = Q \setminus B$

the capability model of Q: A,

and $B \in A$,

then the capability model of P:

$$A - B$$

After hiding the internal events according to the **Rule 2**, the capability model of the entire system becomes the system level capability model. Subsequently, the system capability model is mapped and associated with the task level primitives that define the set of atomic actions that an object is capable of performing.

The computer-aided CSP tools, namely the FDR2 and ProBE [13-14] are then deployed to expedite the process of building capability models, generating the traces, and assisting the formal proof-of -correctness.

3.3 Formal AMHS task primitive repository

Through enumerating the generic components of typical AMHSs, a set of generic capability models for these components are generated. A typical capability description of some combinations of major AMHS components consist of a set of task primitives organized in parallel or in some ordered sequences. These capability models form a database of capabilities building blocks for task/subtask specification and can be used to facilitate the process of task planning.

To illustrate this point, a complex warehousing task may compose of many sub-tasks, and each sub-task containing a sequence of task primitives. By matching these sequences of task primitives with the capability description of a combination of generic MHS components, an AMHS can be built to achieve the specific complex task. As a result, the sequence of actions of performing such complex task can be determined, which provide useful information for subsequent task scheduling. Accordingly, the process of task planning for an AMHS is illustrated in Figure 5.

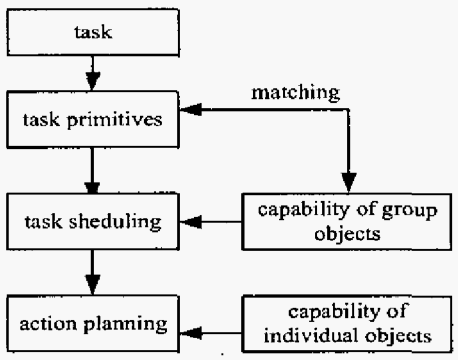

Figure 5: The task planning process

4 Case study

In this section, we demonstrate how the framework is applied to the specification of a typical automated material handling system.

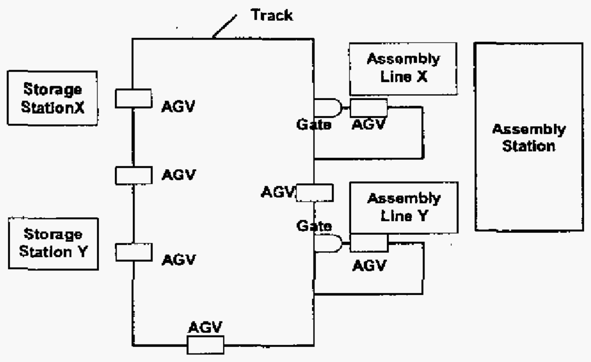

Figure 6: The schematic of the cross-docking warehouse system

Figure 6 shows a cross-docking warehousing system that is supported by a track-based transportation system with seven rail-guided AGVs. A typical delivery task or work order involves the collection of a part from the Storage Station X and moving the part to Assembly Line X by passing through the transfer gate (*Gate 1*). Once an AGV has reached the Assembly Line X, it orders its assigned goods. The Assembly Station then assembles the goods using appropriate tools. The assembled goods is re-loaded onto the AGV from the Assembly Line Y and then brought to the Storage Station Y to be stored.

4.1 Object structure and communication diagrams

The organization and behavior of this cross-docking warehousing system can be succinctly specified using appropriate object structure and object communication diagrams under the proposed framework. These two formal diagrams are constructed by identifying the participating processes, interactions and communications between the components of the warehousing system.

In this system, the active objects identified include *AGVs*, *Gates*, *Assembly Lines*, and *Storages Stations*. In particular, the *AGVs* are the main components within the system that communicate with other components to fulfill work orders. The object structure and the object communication diagrams for this system are given by Figures 7 and 8. According to Figure 7, each *AGV* is operating in an interleaving manner with other *AGVs* whereas the *AGVs* and other subsystems are operating in a parallel manner with the synchronized communications. For example, the *AGV* and the *Gate* are operating in parallel with the synchronized external communication event *enter*. Two specific *Assembly Line* objects, namely *Assembly Line X* and *Assembly Line Y* are being derived from the generic *Assembly Line* object. In particular, *Assembly Line X* is for unloading parts whereas *Assembly Line Y* is for delivering assembled goods. In the object communication diagram (Figure 8) there are four object types, namely, the *Gate*, *AGV*, *Storage Station* and *Assembly Line* with communication channels: *ch0...ch3*. The *Storage Station* is a generic object that represents both types of storage stations for obtaining parts and storing of assembled goods. Therefore, a *Storage Station* can receive *load* and *store* signals. The channels connecting a process with the border of the diagram model the communication with the environment. For example, the *AGV* communicates with the environment through the channel *ch0*, and receives the *task* information from the environment.

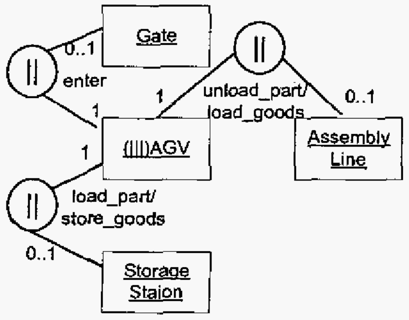

Figure 7: The object structure diagram of the warehousing system

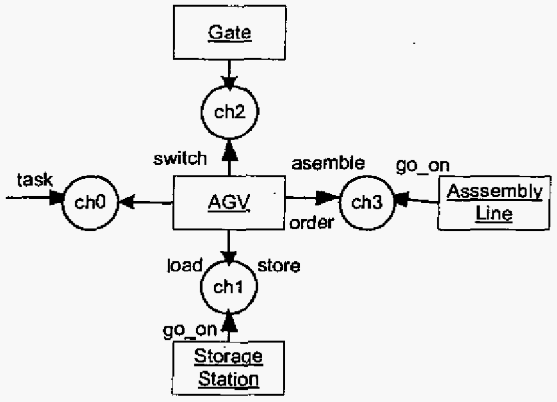

Figure 8: The object communication diagram of the warehousing system

4.2 CSP-based statechart models

For each active object identified by the object structure diagram, a corresponding CSP-based statechart that model its dynamic behaviors is developed. Figure 9 shows the CSP-based statecharts for these objects. The events identified in the behaviors of these objects are described in detail in Appendix A.

The *AGV* statechart (Figure 9a) that outlines the behavior of the *AGV* has two top-level states. Initially, it is in the *waiting* state where it is waiting to accept instructions via *ch0 ? task*. When an event or an instruction is received, it switches to the *active* state. The *active* state is a complex state that defines the behavior of the *AGV*. Under normal operation, an *AGV* performs six sequential operations, namely, *load parts*, *enter the Gate*, *unload parts*, *enter the Gate*, *load goods*, and *store goods*. Under these operations, the *AGV* receives the *go_on* signal from the Storages Station (*ch1 ? go_on*), and Assembly Lines (*ch3 ? go_on*) to continue its actions, and also sends the signals of *assemble* and *order* to the Assembly Lines (*ch3 ! assemble, ch3 ! order*) to instruct the Assembly Line and order the wanted goods. When an *AGV* successfully finishes a task, it then enters the *SKIP* state.

In the case of the *Assembly Line* statechart (Figure 9b), it contains an external choice, as this statechart models the behaviors of both types of assembly lines. When it receives a signal of *assemble*, it then executes the assembly function, and when it receives the *order* signal, it executes the *delivery* function. Similarly, the CSP-based statechart of *Storage Station* (Figure 9c) also contains an external choice, as it models the behaviors of both types of storage stations, which execute the *load part* and *store goods* functions respectively. The behaviors of gate are showed in the Figure 9d, when it receives a *switch* signal, it splits to allow the *AGV* to enter and when an *AGV* has entered, it then *joins*.

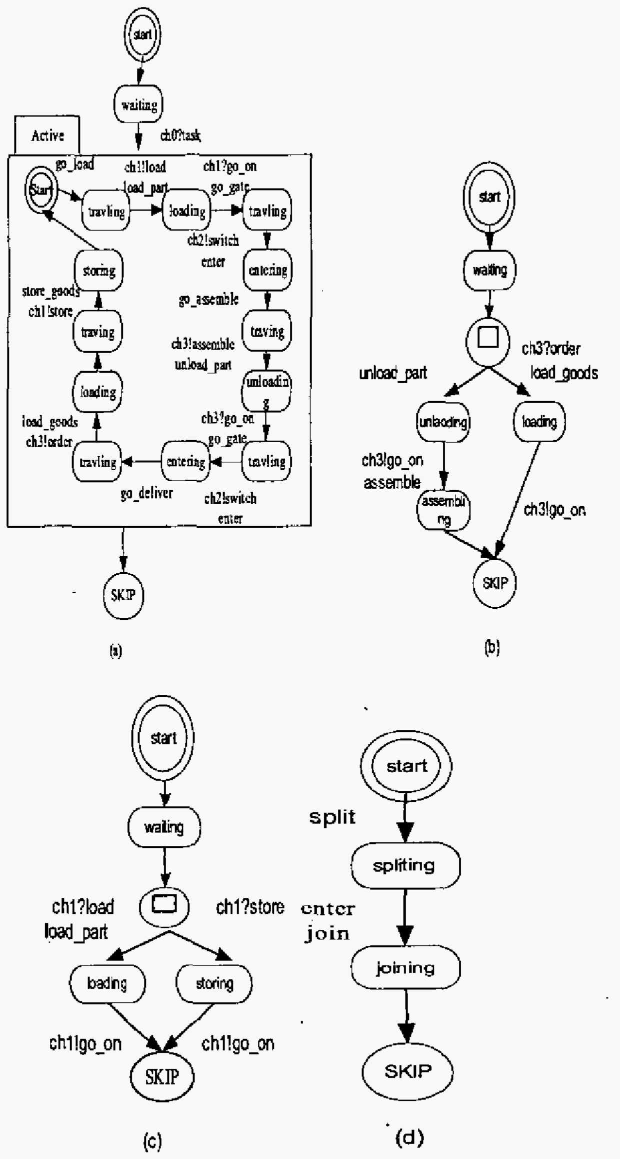

Figure 9: CSP-based statecharts for the *AGV*, *Assembly Line, Storage Station and Gate* object

4.3 Capability models

According to the CSP-based statecharts of the objects that specify the dynamics of the warehousing system, the capability models of each of these objects are then developed:

- Capability model of the *AGV*:

  {ch0 ? task, go_load ,ch1!load, load_part, ch1 ? go_on, go_gate, ch2 ! switch, enter, go_assemble, ch3 ! assemble, unload_part, ch3 ? go_on, go_deliver, ch3 ! order, load_goods, go_store, ,ch1!store, store_goods};

- Capability model of the *Assembly Line*:

  {ch3 ? assemble, unload_part, ch3 ! go_on, assemble}
  {ch3 ? order ,load_goods , ch3 ! go_on};

- Capability model of the *Storage Station*:

  {ch1 ? load, load_part, ch1 ! go_on} *{ch1 ? store, store_goods,*

  ch1 ! go_on};

- Capability model of the *Gate*:

  {ch1 ? switch, split, enter, join}

According to the object diagram of objects, the whole system can be described as *System=AGV* || *Assembly Line* || *Storage Station* || *Gate* in CSP. Based on the **Rule 1** and **Rule 2**, the capability model of the whole system can be built:

{go_load, load_part, go_gate, enter, split, join, go_assemble, unload_part, assemble, go_deliver, load_goods, go_store, store_goods}.

Furthermore, by hiding the event of *go_load, go_gate, enter, split, join, go_assemble, unload_part, go_deliver, load_goods, go_store*, a more abstract capability model is resulted:

{load_part, assemble, store_goods}

These models have been checked by FDR2 and confirmed that there are no pathological problems of deadlock. In addition, the derived capability model that consisting of a set of formal task primitives can be added to the capability repository for subsequent task planning. Characterizing the capabilities of the whole system and its components, the required action plans for specific tasks can be generated with the help of the computer-aided tools. Such plan defines the responsibilities of all AMHS components and their interactions concerned from which the overall task can be achieved, which will be the backbone of the control scheme.

5 Conclusion

In this paper, we have proposed a hybrid framework for task planning of AMHS. Being the central part of this framework, a formal hybrid object-oriented model for specifying the characteristics and behaviors of AMHS has been developed by integrating the notations and semantics of UML and CSP. This integration enables the model to precisely capture the key properties of AMHS, and also enables the model to reason about the pathological problems of deadlock and liveness with the help of CSP model checkers such as FDR2. Meanwhile, the integration of the graphical language of UML with CSP provides a graphical interface to work with CSP that streamlines the specification processes. Based on the hybrid object-oriented model, we have formally described the capabilities of individual components as well as groups of components to build up a database (the repository) so as to facilitate the subsequent generation of action plans. These results will bring us a step closer to the generation of a fully automated task-planning executive for the autonomous control of material handling systems. Moreover, by adopting the capability modeling approach to task planning, dynamic operating conditions can be accommodated, which is extremely important for achieving intelligent, robust and efficient operation for future material handling systems.

6 Acknowledgement

The work described in this paper was supported by the Research Grant Council of the Hong Kong Special Administrative Region, PRC under the CERG Project No. HKU7079/02E.

References

[1] Booch, G, Object-Oriented Analysis and Design with Applications, the Benjamin/Cummings Publishing Company, Inc., California, 1994.

[2] Holt, J, UML for systems engineering-watching the wheels. The institution of electrical engineers.

[3] Hoare, C. A. R, Communicating Sequential Processes, Prentice Hall International, UK, 1985.

[4] Roscoe, A. W., The theory and Practice of Concurrency, Prentice Hall International, UK, 2000.

[5] Winter.K, *Model checking railway interlocking systems*, the 24th Austrial conference on Computer science, 2002, Proc. Vol 4, No.1, pp.~303--310.

[6] Engels. G, Küster. J.M. , and Heckel. R. et al, *A methodology for specifying and analyzing consistency of object-oriented behavioral models*, the 8th European software engineering conference held jointly with 9th ACM SIGSOFT international symposium on foundations of software engineering, 2002, Proc. Vol 26, No. 5, pp. ~186--195.

[7] Robbins. J.E., Medvidovic. N., Redmiles. D.F., and et al, *Integrating architecture description languages with a standard design method*, the 20th internaltional conference on Software engineering, 1998, Proc. pp. ~209--218.

[8] Tsujigado, M., *Application of CSP to the specification description, analysis and software design of elevator control systems and the implementation in Ada*, the IEEE IECON 21st international Conference on Industrial Electronics, Control, and Instrumentation, 1995, Proc. Vol2, pp. ~1555 --1560.

[9] Lau,Y. K. H. and Daniel, R. W., *A CSP model for distributed control software design*, Technical report 1789/89, Oxford University Department of Engineering Science, 1989.

[10] Sun, Y. and Yang, H., *Communication mechanism independent protocol specification based on CSP: a case study*, the 22nd EUROMICRO conference on Hardware and Software Design Strategies, 1996, Proc. pp. ~303--310.

[11] O'donoghue, P. G. and Hull, M. E. C., *Using timed CSP during object oriented design of real-time systems*, Inf. Software Technology, 1996, Proc. Vol.38, No.2, pp. 89 – 102.

[12] Smith, G., *A semantic integration of Object-Z and CSP for the specification of concurrent systems*, FME'97: Industrial Benefit of Formal Methods, Springer-Verlag, 1997.

Appendix A

Table 1: Events of AGV

Event	Description
ch0?task	Receive the task instruction from environment through the channel of ch0
go_load	Move to Storage Station X for loading part
ch1!load	Send the signal of load to Storage Station X
ch1!store	Send the signal of store to Storage Station Y
load_part	Load part from Storage Station X
ch1?go_on	Receive the continue message from the Storage Station X or Y
ch2!switch	Send switch message to the Transfer Gate
go_gate	Move to the Transfer gate
Enter	Enter the Transfer Gate
go_assemble	Move to the Assembly Line X
ch3!assemble	Send assemble instruction to the Assembly Line X
ch3!order	Send order instruction to the Assembly Line Y
unload_part	Unload part to the Assembly Line X
ch3?go_on	Receive the continue message from the Assembly Line X or Y
go_deliver	AGV moves to the Assembly Line Y for loading assembled goods
load_goods	Load goods from the Assembly Line Y
go_store	Move to the Storage Station Y for storing goods
store_goods	Unload the goods to the Storage Station Y to store

Table 2: Events of Assembly Line

Event	Description
ch3?assemble	Receive assemble instruction from AGV
ch3?order	Receive order instruction from AGV
unload_part	Load part from AGV
ch3!go_on	Send the continue message to AGV
load_goods	Unload goods to AGV
assemble	Assemble goods

Table 3: Events of Gate

Event	Description
ch2?switch	Receive switch instruction from AGV
Enter	AGV Enter
split	Open
join	Close

Table 4: Events of Storage Station

Event	Description
ch1?load	Receive the signal of load from AGV
ch1?store	Receive the signal of store from AGV
load_part	Unload part to AGV
ch1!go_on	Send the continue message to AGV
store_goods	Load goods from AGV