

Designing and Modeling Cyberworlds using the Incrementally Modular Abstraction Hierarchy based on Homotopy Theory

Kenji Ohmori · Toshiyasu L. Kunii

Received: date / Accepted: date

Abstract For designing and modeling complicated and sophisticated systems such as cyberworlds, their mathematical foundation is critical. To realize it, two important properties called the homotopy lifting property (HLP) and homotopy extension property (HEP) are applied for designing and modeling a system in a bottom-up way and a top-down way, respectively. In this paper, an enterprise system and a real-time embedded system are considered as important socially emerging cases of cyberworlds, where the π -calculus processes for describing these behaviors formally, a Petri net for explaining process interactions, and XMOS XC programs are modeled and designed by our approach. The spaces in both properties are specified by the incrementally modular abstraction hierarchy by climbing down the abstraction hierarchy from the most abstract homotopy level to the most specific view level, while keeping invariants such as homotopy equivalence and topological equivalence.

Keywords Homotopy Theory · Software Engineering · Cyberworlds · Top-down and Bottom-up Design · Homotopy Lifting/Extension Properties · Pi-Calculus · Event driven · Infinite State Machine

1 Introduction

Cyberworlds are more complicated than any other artificial systems. For designing and modeling cyberworlds, it is necessary to have more powerful foundation than that the current computer science enjoys. As a possible candidate, the incrementally modular abstraction hierarchy (IMAH) has been introduced. Homotopy [Sieradski(1992)],

Kenji Ohmori
Hosei University
3-7-2 Kajinocho, Koganei-shi, Tokyo184-8584 Japan
Tel.: +81-42-387-4546
Fax: +81-42-387-4560
E-mail: ohmori@hosei.ac.jp

Toshiyasu L. Kunii
Morpho, Inc. University of Tokyo, Entrepreneur Plaza 5F
7-3-1 Hongo, Bunkyo-ku, Tokyo 113-0033
E-mail: kunii@ieee.org

[Spanier(1966)] constituting the most abstract level of the IMAH has two important properties. These properties are called the homotopy lifting property (HLP) and the homotopy extension property (HEP). In mathematics, the HLP lifts projection to homotopy and the HEP extends inclusion to homotopy. When applying these properties in computer science, the HLP and the HEP are applicable to modeling and designing systems in a bottom-up way and a top-down way, respectively. When designing and modeling a complicated system, it is started from a certain point where the top-down way is applied to decomposing it and the bottom-up way is to composing it. In other words, the HLP and the HEP constitute another hierarchy which is called the component hierarchy. The HLP constructs the whole from its components and the HPL divides the whole into its components. When using the HLP and the HEP, an invariant as homotopy equivalence is preserved. The invariant preserving gives strong foundation in designing and modeling sophisticated and complicated cyberworlds as done in classical physical worlds by mass and energy as invariants.

The IMAH consists of 7 levels. These are the homotopy level, the set theoretical level, the topological space level, the adjunction space level, the cellular space level, the presentation level and the view level. By climbing down the abstraction hierarchy, a developing or modeling system is transformed from general concepts to specific entities. At the homotopy level, the HLP and the HEP are utilized for composing or decomposing a given system, respectively. At the set theoretical level, the spaces defined at the homotopy level are specified by giving the structure of a set with elements and subsets. At the topological space level, sets are accommodated by topological properties. As computer science deals with discrete entities, algebraic topology is used at this level. At the adjunction space level, two subspaces in different disjoint spaces are attached if these are identified as equivalent. At the cellular space level, an abstract physical structure is given. The presentation level and the view level depend on an application field. If it is an object oriented system, the presentation level is described by templates or classes and the view level by instances.

The IMAH has been introduced in paper [Kunii(2005)]. Then, the HLP and the HEP have been applied to the field of computer science in papers [Kunii and Ohmori(2006)], [Ohmori and Kunii(2006)], [Ohmori and Kunii(2007a)], [Ohmori and Kunii(2007b)]. The π -calculus [Milner(1999)], [Sangiorgi and Walker(1999)], [Hennessy(2001)] and the Petri net are important concepts when discussing a complicated and sophisticated system. These ideas have been applied to constituting fundamental theory of business process modeling [Havey(2005)]. The designing of π -calculus processes have been discussed in papers [Ohmori and Kunii(2008a)], [Ohmori and Kunii(2008b)], [Ohmori and Kunii(2009)]. In this paper, an enterprise system and a real-time embedded system are modeled and designed. Using the HLP in a bottom-up way, the π -calculus processes are modeled for both systems by constructing the whole from its components. Using the HEP in a top-down way, the Petri net for the enterprise system and XMOS XC programs [May(2007)] for the embedded system are designed by dividing the whole into its components. Each space constituting the HEP or the HLP is specified by climbing down the abstraction hierarchy while keeping equivalences as invariants. Through this paper, mathematical foundation for designing and modeling the enterprise system is clarified. As these systems are rich in distributed communication processes, our approach for these systems is applicable to designing and modeling cyberworld systems inevitably marked by these characteristics.

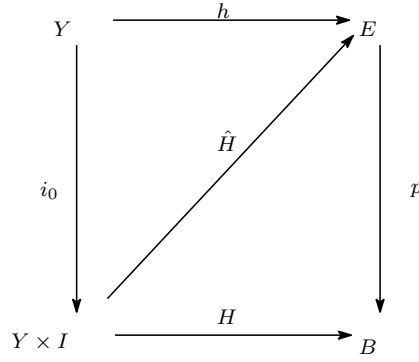


Fig. 1 Homotopy Lifting Property

2 Preparation

2.1 The homotopy lifting property and homotopy extension property

The mathematical backgrounds for the HLP and HEP are summarized as follows.

- Continuous maps p, q are homotopic if there exists a continuous map $H : X \times I \rightarrow Y$ such that $H(x, 0) = p(x)$ and $H(x, 1) = q(x)$, where I is the unit interval $[0, 1]$. H is called homotopy of p and q , denoted by $p \simeq q$.
- A continuous map $\lambda : I \rightarrow X$ yields a path. $\lambda(0) = x$ and $\lambda(1) = y$ are called the initial and terminal points. The path is denoted by $w = (W, \lambda)$ where $W = \lambda(I)$.
- A fiber bundle is a quadruple $\xi = (E, B, F, p)$ consisting of a total space E , a base space B , a fiber F , and a bundle projection that is a continuous surjection called F -bundle $p : E \rightarrow B$ such that there exists an open covering $\mathcal{U} = \{U\}$ of B and, for each $U \in \mathcal{U}$, a homeomorphism called a coordinate chart $\varphi_U : U \times F \rightarrow p^{-1}(U)$ exists such that the composite $U \times F \rightarrow p^{-1}(U) \rightarrow U$ is the projection to the first factor U . Thus the bundle projection $p : E \rightarrow B$ and the projection $p_B : B \times F \rightarrow B$ are locally equivalent. The fiber over $b \in B$ is defined to be equal to $p^{-1}(b)$, and it is noted that F is homeomorphic to $p^{-1}(b)$ for every $b \in B$, namely $\forall b \in B, F \cong p^{-1}(b)$.
- Given any commutative diagram of continuous maps as shown in Fig. 1, the map $p : E \rightarrow B$ has the homotopy lifting property if there is a continuous map $\hat{H} : Y \times I \rightarrow E$ such that $\hat{H} \times i_0 = h$ and $p \circ \hat{H} = H$. The homotopy \hat{H} thus lifts H through p and extends h over i_0 where $i_0(a) = (a, 0)$.
- A fibration is a continuous map $p : E \rightarrow B$ that has the homotopy lifting property. The homotopy extension property is dual to the homotopy lifting property. The homotopy extension property is defined as follows.
- Given any commutative diagram of continuous maps as shown in Fig. 2, there is a continuous map $\hat{K} : X \rightarrow Y^I$ such that $p_0 \times \hat{K} = k$ and $\hat{K} \times i = K$. The homotopy \hat{K} thus extends K over i and lifts k through p_0 where $p_0(\lambda) = \lambda(0)$.
- An inclusion of a closed subspace $i : A \hookrightarrow X$ is a cofibration if i has the homotopy extension property. Y^I is the path space on Y . The path space is defined as follows.
- The path space on X , denoted X^I , is the space $\{\lambda : I \rightarrow X | \text{continuous}\}$ endowed with the compact-open topology.

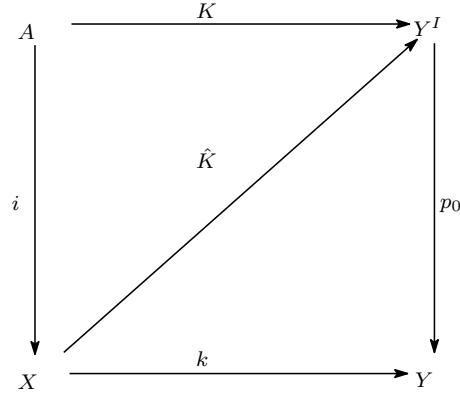


Fig. 2 Homotopy Extension Property

- Let us start with a topological space X and attach another topological space Y to it. Then, $Y_f = Y \sqcup_f X = Y \sqcup X / \sim$ is an adjunction space obtained by adjoining Y to X by an adjunction map, f (or by identifying each point $y \in Y_0 \mid Y_0 \subset Y$ with its image $f(y) \in X$ by a continuous map f). \sqcup denotes a disjoint union. The adjunction map f is a continuous map such that $f : Y_0 \rightarrow X$, where $Y_0 \subset Y$. Thus, the adjunction space $Y_f = Y \sqcup X / \sim$ is a case of quotient spaces $Y \sqcup X / \sim = Y \sqcup_f X = Y \sqcup X / (y \sim f(y) \mid \forall y \in Y_0)$.

2.2 The π -calculus process

The π -calculus is a parallel processing model as a case of process calculus where communication links are dynamically changed. The action prefixes π are the generalization of actions. An action prefix expresses either sending or receiving a message or making a silent transition. The π -prefixes are defined as follows.

$$\begin{aligned} \pi &::= x(y) \text{ receive } y \text{ along } x \\ &::= \bar{x}(y) \text{ send } y \text{ along } x \\ &::= \tau \text{ unobservable action} \end{aligned}$$

The set P^π of π -calculus process expressions is defined as follows:

$$P ::= 0 \mid \sum_{\lambda \in A} \pi_\lambda . P_\lambda \mid P_1 \mid P_2 \mid \text{new } a P \mid !P$$

0 is an inaction process that can do nothing. The processes $\sum_{\lambda \in A} \pi_\lambda . P_\lambda$ are called sums.

Each item is a process and only one item is executed. A is any finite indexing set. In a sum $\sum_{\lambda \in A} \pi_\lambda . P_\lambda$, it is said that P_λ is guarded by π_λ since the action by π_λ has to proceed before P_λ becomes active. $P_1 \mid P_2$ can proceed independently and interact by shared names. $\text{new } a P$ restricts the scope of the name a to P . $!P$ repeats P infinitely.

3 Design and Modeling of Internet Company as a Model of Socially Emerging Enterprise Systems

By assuming a small company, we design and model company's activities using the HLP and the HEP as well as the IMAH. The small company, named Internet Company, operates an Internet business. The company has Editorial, Service and Accounting Departments. Each department has its own business model. Interactions among departments constitute company business model.

Editorial Department has the following business model. The department operates department works and several projects as its business. The department is managed by a department manager and each project is controlled by a project leader under the department manager. Mike, staff of the department, can engage in a department work or a project. When he wants to join a project, he has to talk to the department manager on his intention of movement. Then, the department manager talks Mike's joining to the project leader. Mike is also allowed to switch projects or return to a department work by the same procedure as at his joining a project.

3.1 Using the Homotopy Lifting Property

The department business model is formally clarified using the π -calculus. The π -calculus processes are designed in a bottom-up way using the HLP as shown in Fig. 1, where Y, B and E describe the set of states, a state transition diagram and π -calculus processes, respectively.

3.1.1 Modeling the Department Business Model

Before constructing the HLP, the space of the department business model, denoted by S^d , is defined at the set theoretical level and the topological space level.

The department business model is essentially constructed by 1) businesses consisting of department works and projects, 2) movements causing a business change of Mike and 3) agents performing the department business. Therefore, S^d consists of the sets of businesses, movements and agents, denoted by S_b^d, S_m^d and S_a^d , respectively. S_b^d consists of two elements $e_{b_0}^d$ and $e_{b_1}^d$ for 'doing a department work' and 'doing a project', respectively. S_m^d consists of three elements $e_{m_0}^d, e_{m_1}^d$, and $e_{m_2}^d$ for 'join', 'drop' and 'switch', respectively. S_a^d consists of three elements $e_{a_0}^d, e_{a_1}^d$, and $e_{a_2}^d$ for 'Mike', 'department manager' and 'project leaders', respectively. As there are multiple projects in the department, $e_{a_2}^d$ is the set of project leaders. For example, if there are two projects, each of which has a project leader, denoted by $e_{a_{2a}}^d$ and $e_{a_{2b}}^d$, $e_{a_2}^d = \{e_{a_{2a}}^d, e_{a_{2b}}^d\}$. The topological spaces for these sets are then given by defining the discrete topology on each set.

3.1.2 Constructing a State Transition Diagram at the Cellular Space Level

A state transition diagram is clarified for constructing space B . The state transition diagram is common to all agents. It is constructed in a formal way starting from the adjunction space level. According to the conventional notation of a state transition diagram, a state and an event are described by a node and an arrow, respectively.

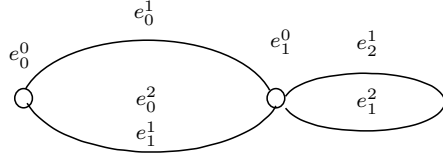


Fig. 3 State Transition Diagram

Each business and movement are described as a state and an event. $e_{b_0}^d$ and $e_{b_1}^d$ are represented by 0-dimensional points e_0^0 and e_1^0 , respectively. $e_{m_0}^d, e_{m_1}^d$, and $e_{m_2}^d$ are represented by 1-dimensional edges e_0^1, e_1^1 and e_2^1 , respectively. 0-dimensional points and 1-dimensional edges are topologically equivalent to 0-dimensional and 1-dimensional open balls, respectively.

Using filtration, the points and the edges are connected for forming a CW-complex in such a way that the boundaries of edges e_0^1 and e_1^1 are attached to points e_0^0 and e_1^0 and the boundaries of edge e_2^1 are attached to point e_0^0 as shown in Fig. 3 using adjoining functions $f_0 : \partial e_0^1 \rightarrow e_0^0, e_1^0, f_1 : \partial e_1^1 \rightarrow e_0^0, e_1^0$ and $f_2 : \partial e_2^1 \rightarrow e_1^0, e_1^0$ where ∂e is a boundary of e . Each adjoining function yields an adjunction space in such a way that $\{e_0^1\} \sqcup \{e_0^0, e_1^0\} / \{x \sim f_0(x) | x \in \partial e_0^1\}$.

2-dimensional surfaces e_0^2 and e_1^2 , topologically equivalent to 2-dimensional open balls, and 3-dimensional sphere e_0^3 , topologically equivalent to a 3-dimensional open ball, are introduced to construct the state transition diagram as a cellular space. Boundaries of e_0^2 and e_1^2 are attached to the edges as shown in Fig. 3 and boundaries of e_0^3 are attached to e_0^2 and e_1^2 . Any state transition diagram is represented as a sphere if it is a connected graph.

3.1.3 Constructing Businesses and an Interval at the Topological Space Level

$Y \times I$ is constructed as follows. Homotopy $H : Y \times I \rightarrow B$ shows how the map from Y to B changes continuously along I . I is an interval $[0, 1]$. It is interpreted as continuous changes of time, space or events, or changes of something. In a discrete system, I is the sequence of sampled points along changes of something. Y and B are the set of businesses and the state transition diagram, respectively. If I is described by a sequence of movements for describing business changes, then H shows how the current business is changed by I . I is defined by $(e_{m_0}^d | e_{m_1}^d | e_{m_2}^d)^*$, where any sequence of movements is generated by a sequence of $e_{m_0}^d, e_{m_1}^d$ and $e_{m_2}^d$. The shortest sequence of movements is a single movement.

Homotopy H is defined by $H(s, t) = e$ where $s \in S_b^d, t \in S_m^d$ and $e \in \{e_0^1, e_1^1, e_2^1\}$ where e is the edge for movement t occurring at business s . If (s, t) does not have any corresponding edge, then it is mapped to ϕ that means t is an unacceptable movement at s . For example, $H(e_{b_0}^d, e_{m_0}^d) = e_0^1$ and $H(e_{b_1}^d, e_{m_0}^d) = \phi$.

3.1.4 Constructing π -calculus Processes

Space E for π -calculus processes is defined using B and $Y \times I$. At first, the state transition diagram of B is copied to E for each agent to obtain its agent process diagram at the cellular space level. The agent process diagrams are then transformed

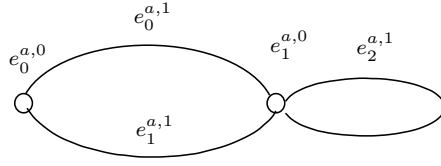


Fig. 4 Agent Process Diagram

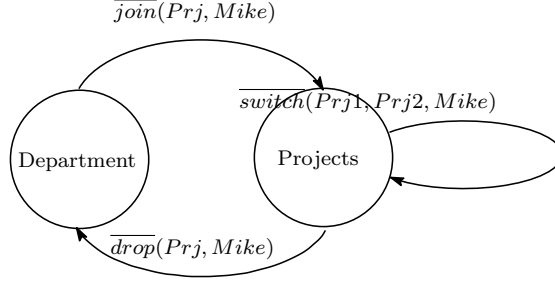


Fig. 5 Agent Process Diagram for Mike

into diagrams at the presentation level. Finally the diagrams are represented at the view level by specifying the project of the department.

1. The Cellular Space Level

To create E for π -calculus processes, a copy of the state transition diagram is provided for each agent as its agent process diagram as shown in Fig. 4 where a is m for Mike, d for the department manager or p for the project leaders. Mathematically, it is defined by $p^{-1}(e_j^i) = \{e_j^{m,i}, e_j^{d,i}, e_j^{p,i}\}$. From this definition, it is clear that $H = p \circ \hat{H}$. Each diagram lies at the cellular space level.

2. The Presentation Level

Each movement is composed a sequence of actions. For example, if Mike joins a project from a department work, the actions for $(e_{b_0}^c, e_{m_0}^c) \in Y \times I$ is expresses by the following sequence.

- y_{j_0} : Mike notifies the department manager of his join to a project.
- y_{j_1} : The department manager receives the message of the project join from Mike.
- y_{j_2} : The department manager adds Mike to the project.
- y_{j_3} : The project leader receives the message of Mike's join from the department manager.

These actions are mapped by \hat{H} and used to express transitions of agent process diagrams by π -prefixes. As $y_{j_0}, y_{j_1}, y_{j_2}$ and y_{j_3} correspond to $e_0^{m,1}, e_0^{d,1}, e_0^{p,1}$ and $e_0^{p,1}$, these are replaced by $\overline{join}(Prj, Mike), \overline{join}(prj, stf), \overline{prj}(Add, stf)$ and $\overline{prj}(msg, stf)$, respectively. By transforming the other actions to π -prefixes, the agent process diagrams are obtained at the presentation level as shown in Fig. 5, 6 and 7, where an arrow is added to each edge to indicate the direction of a transition. Fig. 5 shows how Mike behaves. When he is doing a department work, he can move to one of projects by sending a message through $join$ to the department manager. The message specifies in which project he wants to join. When he is doing some project, he can switch to another project by sending a message through

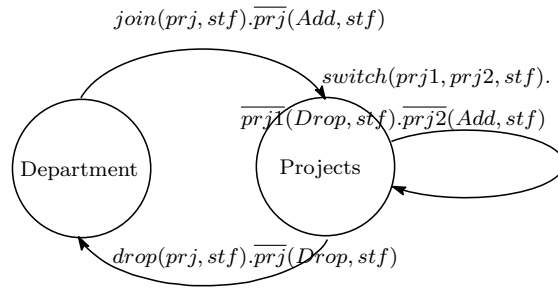


Fig. 6 Agent Process Diagram for Department Manager

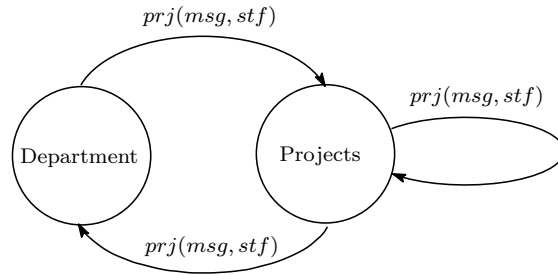


Fig. 7 Agent Process Diagram for Project Leader

switch to the department manager. The message specifies the source project and the destination project. He can also return to a department work by sending a message through *drop* to the department manager by specifying from which project he wants to return.

Fig. 6 shows how the department manager behaves. When he receives a message from Mike through *join* for moving from a department work to a project, he send a message through *prj* to the project leader that Mike is added to his project. When he receives a message from Mike through *switch* for switching to another project, denoted by *prj2*, while Mike is doing some project, denoted by *prj1*, he sends a message to the current project leader through *prj1* by specifying that Mike is dropped from the current project. Then, he sends another message to the next project leader through *prj2* by specifying that Mike is added to this project. When he receives a message from Mike through *drop* for returning to a department work, he sends a message drop to the project leader through *prj* for specifying from which project he leaves.

Fig. 7 shows how a project leader behaves. When Mike moves to a project from a department work, the project leader receives a message from the department manager through *prj*. When Mike switches projects, both project leaders receive a message from the department manager through *prj*. When Mike returns to a department work from the current project, the project leader receives a message from the department manager.

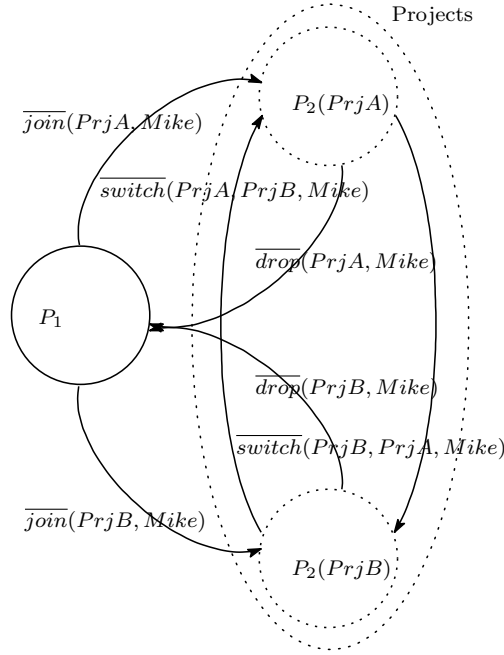


Fig. 8 Agent Process Diagram for Mike at View Level

By climbing down the IMAH from the cellular space level to the presentation level, the properties of the agent process diagrams are preserved. New invariants specifying messages for each movement are added at the presentation level.

3. The View Level

At the presentation level, it is not clear which project Mike is joining. At the view level, the projects are explicitly clarified in the agent process diagram for Mike as shown in Fig. 8, where it is assumed that there are two projects $PrjA$ and $PrjB$. In Fig. 8, each node is replaced by a process in such a way that processes P_1 and P_2 are provided for "Department" and "Project". Each project is specified by an argument of P_2 . When Mike moves from a department work to $PrjA$, P_1 is denoted by the conjunction of the process for movement, which is shown by a π -prefix, and the destination process P_2 in a such a way that $P_1 = \overline{join}(PrjA, Mike).P_2(PrjA)$. As Mike can move to $PrjB$, too, the general expression is obtained as follows.

$$P_1 = \sum_{prj \in \{PrjA, PrjB\}} \overline{join}(prj, Mike).P_2(prj).$$

When Mike is doing a project for $PrjA$, he can switch to another project for $PrjB$ or return to a department work. As $P_2(PrjA)$ is denoted by $\overline{switch}(PrjA, PrjB, stf).P_2(PrjB)$ or $\overline{drop}(PrjA, stf).P_1$, the following expression is obtained.

$$P_2(PrjA) = \overline{switch}(PrjA, PrjB, stf).P_2(PrjB)$$

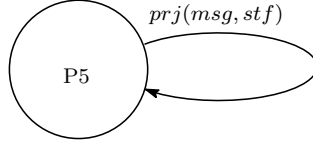


Fig. 9 Agent Process Diagram for Project Leaders at View Level

$$+\overline{drop}(PrjA, stf).P_1.$$

In the same way, the following expression is obtained for $P_2(PrjB)$.

$$P_2(PrjB) = \overline{switch}(PrjB, PrjA, stf).P_2(PrjA) \\ +\overline{drop}(PrjB, stf).P_1.$$

The agent process diagram for the department manager is also described at the view level and the π -calculus processes are obtained as follows.

$$P_3 = join(prj, stf)\overline{prj}(Add, stf).P_4.$$

$$P_4 = switch(prj1, prj2, stf). \\ \overline{prj1}(Drop, stf).\overline{prj2}(Add, stf).P_4 \\ +drop(prj, stf)\overline{prj}(Drop, stf).P_3.$$

The state minimization algorithm for the finite state machine is applied to the agent process diagram for the project leaders. As all states are recognized as an identical state, the agent process diagram for the project leaders is obtained as shown in Fig. 9. The π -calculus process is obtained from the minimized diagram.

$$P_5 = prj(msg, stf).P_5.$$

3.2 Using the Homotopy Extension Property

Internet Company has Editorial, Service and Accounting Departments. An editor of Editorial Department composes a Web document and brings it to a customer by business trip. Expenses by business trip is transacted by Accounting Department through a service person of Service Department. By receiving it, the service person formats it to a journal slip and hands the journal slip to an accountant of Accounting Department. The accountant records it to the ledger system. It is assumed that Editorial Department has more employees than Service Department and Accounting Department has only one accountant. This assumption does not make Accounting Department necessitate mutual exclusion when recording a journal slip.

Using the homotopy extension property, a Petri-net is created for the company business model. The homotopy extension property is shown in Fig. 2. The company business model is constituted by X as the workflows, A as the tasks by a single editor, service person and accountant, Y as a Petri-net and Y^I as a set of path functions. These spaces are obtained as follows.

3.2.1 The Workflow of Internet Company

Editor e_p at Editorial Department performs the following actions repeatedly.

$x_0^{e_p}$: e_p is ready for being assigned to a new task.

$x_1^{e_p}$: e_p is editing a document.

$x_2^{e_p}$: e_p is ready for visiting a customer.

$x_3^{e_p}$: e_p is handing a business trip application to a service person.

The editor task is described by $S^e = \{x_i^{e_p} | i = 0..3, p = 0..m-1\}$.

Service person t_q of Service Department performs the following actions repeatedly.

$x_0^{t_q}$: t_q is available for an editor.

$x_1^{t_q}$: t_q is receiving a business trip application from an editor.

$x_2^{t_q}$: t_q is waiting for the accountant.

$x_3^{t_q}$: After formatting the application to a journal slip, t_q is handing it to the accountant.

The service person task is described by $S^t = \{x_i^{t_q} | i = 0..3, q = 0..n-1\}$, where $x_1^{t_q}$ is synchronized with $x_3^{e_p}$ for some e_p .

Accountant a operates the following actions.

x_0^a : a is available for a service person.

x_1^a : a is receiving a journal slip.

x_2^a : a is ready for accounting process.

x_3^a : a is recording the journal slip to the general ledger.

The accountant task is described by $S^a = \{x_0^a, x_1^a, x_2^a, x_3^a\}$, where x_1^a is synchronized with $x_3^{t_q}$ for some t_q .

Discrete topological spaces T^e , T^a and T^t are introduced on S^e , S^a and S^t .

The workflow of Internet Company is described by $T^w = T^e \sqcup_f T^a \sqcup_f T^t$ where f is an adjuncting function and identifies $x_3^{e_p}$ with $x_1^{t_q}$ and $x_3^{t_q}$ with x_1^a . The workflow constitutes X of the HEP and the tasks of a single editor, service person and accountant compose A .

3.2.2 Obtaining the Petri Net

Y is the space of a Petri net and Y^I is the space of the trajectories for tokens of the Petri net. At first, K is defined for mapping from A to Y^I . Assuming that A consists of the tasks of editor e_0 , service person t_0 and accountant a . Editor e_p has already handed i_p

applications and service person t_q has received j_q applications where $\sum_{p=0}^{m-1} i_p = \sum_{q=0}^{n-1} j_q$.

Service person t_q has already handed k_q journal slips and accountant a has recorded l journal slips where $\sum_{q=0}^{n-1} k_q = l$ and $j_q = k_q$ or $j_q = k_q + 1$. When $j_q = k_q + 1$, the journal slip is waiting for the accountant until the accountant is available.

Then, it is assumed that e_0 , t_0 and a perform the following actions.

- Editor e_0 achieves the $(i_0 + 1)$ th task consisting of $x_0^{e_0, i_0+1}$, $x_1^{e_0, i_0+1}$, $x_2^{e_0, i_0+1}$ and $x_3^{e_0, i_0+1}$.
- service person t_0 does the $(j_0 + 1)$ th task of $x_0^{t_0, j_0+1}$, $x_1^{t_0, j_0+1}$, $x_2^{t_0, j_0+1}$ and $x_3^{t_0, j_0+1}$, where $x_3^{e_0, i_0+1}$ is synchronized with $x_1^{t_0, j_0+1}$.

- Accountant a does $x_0^{a,l+\alpha}, x_1^{a,l+\alpha}, x_2^{a,l+\alpha}$ and $x_3^{a,l+\alpha}$, where $\alpha = 1$ and $x_1^{a,l+\alpha}$ is synchronized with $x_3^{t_0,j_0+1}$. When $\alpha > 1$, other applications, originated from other than e_0 , overtake $x_3^{e_0,i_i+1}$ and is recorded earlier than $x_1^{a,l+\alpha}$.

As e_0, t_0 and a have a sequence of actions which are up to the (i_0+1) th application and $(l+\alpha)$ th journal slips, trajectories of actions are defined to show how activities of them have been processed. Therefore, path functions $\lambda^{e_0}, \lambda^{t_0}$ and λ^a are obtained. The path functions map interval I to the trajectories generated by e_0, t_0 and a . A simple example is given to show how trajectories are obtained from an interval.

3.2.3 A Simple Case

Assuming that $m = 3$ and $n = 2$ in the environment that an editor does not overtake another editor and a service person does not either. The trajectories of e_0, t_0 and a are shown in Fig. 10 where e_0 has handed 3 applications, t_0 has received 4 applications and has handed 4 journal slips and a has recorded 7 journal slips. e_0 is synchronized at $x_3^{e_0,0}$ and $x_3^{e_0,2}$ with t_0 at $x_1^{t_0,0}$ and $x_1^{t_0,3}$. t_0 and a are synchronized each other at $x_3^{t_0,j}$ and $x_3^{a,2j}$ where $j = 0, 1, 2, 3$. When two tasks are synchronized, the trajectories touch each other at the synchronized actions as shown in Fig. 10. Interval I is defined as time until 3 applications of e_0 , 4 journal slips of t_0 and 7 journal slips of a have been processed.

1. Defining K

- For e_0 , K is defined to map $x_0^{e_0}$ to λ^{e_0} and $x_1^{e_0}, x_2^{e_0}, x_3^{e_0}$ to ϕ . I is given by a sequence of time when events related to e_0 occurs such that

$$I = [i_0^{e_0,0}, i_1^{e_0,0}, i_2^{e_0,0}, i_3^{e_0,0}, \dots, i_0^{e_0,2}, i_1^{e_0,2}, i_2^{e_0,2}, i_3^{e_0,2}]$$
where $x_j^{e_0,k}$ occurs at $i_j^{e_0,k}$. Therefore,

$$K(x_0^{e_0})(i_j^{e_0,k}) = \lambda^{e_0}(i_j^{e_0,k}) = x_j^{e_0,k}.$$
- For t_0 , K is defined to map $x_0^{t_0}$ to λ^{t_0} and $x_1^{t_0}, x_2^{t_0}, x_3^{t_0}$ to ϕ . I is given by a sequence of time when events related to t_0 occurs such that

$$I = [i_0^{t_0,0}, i_1^{t_0,0}, i_2^{t_0,0}, i_3^{t_0,0}, \dots, i_0^{t_0,3}, i_1^{t_0,3}, i_2^{t_0,3}, i_3^{t_0,3}]$$
where $x_j^{t_0,k}$ occurs at $i_j^{t_0,k}$, and $i_1^{t_0,0} = i_1^{e_0,0}$ and $i_3^{t_0,3} = i_3^{e_0,2}$ for synchronization. Therefore,

$$K(x_0^{t_0})(i_j^{t_0,k}) = \lambda^{t_0}(i_j^{t_0,k}) = x_j^{t_0,k}.$$
- For a , K is defined to map x_0^a to λ^a and x_1^a, x_2^a, x_3^a to ϕ . I is given by a sequence of time when events related to a occurs such that

$$I = [i_0^{a,0}, i_1^{a,0}, i_2^{a,0}, i_3^{a,0}, \dots, i_0^{a,6}, i_1^{a,6}, i_2^{a,6}, i_3^{a,6}]$$
where $x_j^{a,k}$ occurs at $i_j^{a,k}$ and $i_3^{a,j} = i_1^{a,2j}$ for synchronization. Therefore,

$$K(x_0^a)(i_j^{a,k}) = \lambda^a(i_j^{a,k}) = x_j^{a,k}.$$

2. Defining \hat{K}

- For e_p , \hat{K} is defined to map $x_0^{e_p}$ to λ^{e_p} and $x_1^{e_p}, x_2^{e_p}, x_3^{e_p}$ to ϕ where $p = 0, 1, \dots, m-1$. I is given by a sequence of time when events related to the editors occur such that

$$I = [\begin{array}{l} i_0^{e_0,0}, i_1^{e_0,0}, i_2^{e_0,0}, i_3^{e_0,0}, \\ i_0^{e_1,0}, i_1^{e_1,0}, i_2^{e_1,0}, i_3^{e_1,0}, \end{array}]$$

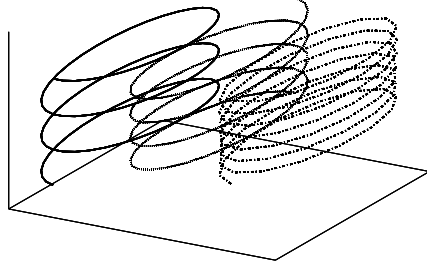


Fig. 10 Trajectories of Actions

$$\begin{aligned}
 & \dots\dots\dots \\
 & i_0^{e_{m-1,0}}, i_1^{e_{m-1,0}}, i_2^{e_{m-1,0}}, i_3^{e_{m-1,0}}, \\
 & i_0^{e_{0,1}}, i_1^{e_{0,1}}, i_2^{e_{0,1}}, i_3^{e_{0,1}} \\
 & \dots\dots\dots \\
 & \dots\dots\dots \\
 & i_0^{e_{m-1,5}}, i_1^{e_{m-1,5}}, i_2^{e_{m-1,5}}, i_3^{e_{0m-1,5}}, \\
 & i_0^{e_{0,6}}, i_1^{e_{0,6}}, i_2^{e_{0,6}}, i_3^{e_{0,6}} \\
 &]
 \end{aligned}$$

where $x_j^{e_p,k}$ occurs at $i_j^{e_p,k}$. Therefore,

$$\hat{K}(x_0^{e_p})(i_j^{e_p,k}) = \lambda^{e_p}(i_j^{e_p,k}) = x_j^{e_p,k}$$

$$\hat{K}(x_0^{e_p})(i_j^{e_{p'},k}) = \phi \text{ when } p \neq p'.$$

– For t_q , λ^{t_q} and \hat{K} are also defined in the same way.

It is clear from these definitions that $K = \hat{K} \circ i$.

3. Constructing Y at the Adjunction Space Level

Y constituting a template of the Petri net is obtained by projecting Y^I such that $x_i^{e_p,j_0}, x_i^{t_q,j_1}$ and x_i^{a,j_2} are mapped to x_i^e, x_i^t and x_i^a , respectively, as shown in Fig. 11. The template is described at the adjunction space level.

4. Constructing Y at the Presentation Level

By providing places equivalent to 0-dimensional open balls and transitions equivalent to 1-dimensional open balls, Y is constituted at the presentation level through the cellular space level.

5. Constructing Y at the View Level

$p_0 : Y^I \rightarrow Y$ is defined to map the initial point of each trajectory to a token of the Petri net. At the view level, tokens corresponding to the initial points of the trajectories are also provided. $x_0^{e_p,0}, x_0^{t_q,0}$ and $x_0^{a,0}$ are mapped to x_0^e, x_0^t and x_0^a , respectively, as shown in Fig. 12.

Finally, k which maps X to Y is defined as follows.

For e_p ,

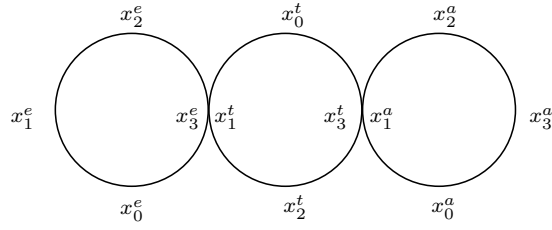


Fig. 11 Template of Petri Net

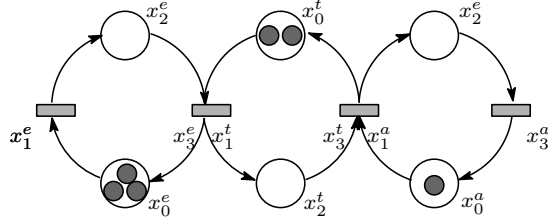


Fig. 12 Petri Net for Internet Company

$$k(x_0^{e,p}) = x_0^e,$$

$$k(x_i^{e,p}) = \phi \text{ when } i = 1, 2, 3.$$

For t_q ,

$$k(x_0^{t,q}) = x_0^t,$$

$$k(x_i^{t,q}) = \phi \text{ when } i = 1, 2, 3.$$

For a ,

$$k(x_0^{a,q}) = x_0^a,$$

$$k(x_i^{a,q}) = \phi \text{ when } i = 1, 2, 3.$$

Therefore, it is clear that $k = p_0 \circ \hat{K}$.

3.2.4 The General Case

The Petri net for the workflow of Internet Company is obtained by generalizing the simple case. For each agent, a spiral line is provided to show the trajectory of agent activities. These spiral lines give Y^I at the topological space level. As an agent is synchronized with another agent, the synchronization is given by the tangent point of the two spiral lines. These jointed spiral lines give Y^I to Y such that $x_i^{e,p,j}, x_i^{t,q,j}, x_i^{a,j}$ to x_i^e, x_i^t, x_i^a at the adjunction space level. By projecting Y^I in the same way as the simple case, Y representing a template of the Petri net for Internet Company is obtained at the adjunction space level. By providing places and transitions for the template, Y is given at the presentation level. By mapping the initial point of each trajectory to Y as a token, Y is given in the form of the Petri-net with tokens at the view level.

4 Car Robot as an Emerging Case of Real-time Embedded Systems

For a toy robot called Car Robot, the control program is designed using the HLP and the HEP as well as the IMAH. Car Robot has the brain, the two sensors and the two wheels. Car Robot behaves as follows: 1) Car Robot runs along a black line so that the black line lies between the two wheels, 2) the brain can control the speed of a wheel, 3) immediately after the right (left) sensor finds that the right (left) wheel touches the black line, the right (left) wheel is automatically stopped in order to turn Car Robot to the right (left) for returning the normal route without receiving a command from the brain, 4) later, the brain may send commands to a wheel to change its speed.

4.1 Using the Homotopy Lifting Property

The processes of Car Robot is described using the π -calculus. As described before, π -calculus processes are composed in a bottom-up way using the HLP, where Y, I, B and E describe the Car Robot behaviors, events, a state transition diagram and π -calculus processes, respectively.

4.1.1 Constructing the State Transition Diagram

1. The set theoretical space level

As described before, Car Robot is defined at the set theoretical level. As the space of *carrobot* consists of 1) *behaviors* to show in which situation Car Robot behaves, 2) *events* to indicate which action changes Car Robot situation and 3) *agents* to show which entity causes events, it is described as $carrobot = \{behaviors, events, agents\}$. As there are two situations, it is described by $behaviors = \{normal, derailment\}$. As crossing the black line and changing speed of Car Robot cause situation change, *events* are composed by $events = \{lines, speeds\}$. As crossing line has two different situations: touching the line or separating from the line, *lines* are described by $lines = \{touch, separate\}$. As there are three different speeds to control Car Robot, *speeds* are defined by $speeds = \{fast, medium, slow\}$. As there are four different kind of agents, *agents* are described by $agents = \{brain, sensors, wheel_controllers, wheels\}$. Each agent has the following instances.

$$\begin{aligned} sensors &= \{left_sensor, right_sensor\}. \\ wheels &= \{left_wheel, right_wheel\}. \\ wheel_controllers &= \{left_wheel_controller, \\ &\quad right_wheel_controller\}. \end{aligned}$$

2. The cellular space level

The state transition diagram for Car Robot is defined as follows. *behaviors* and *events* are used as states and transitions. Using the same procedure as for the department business model, Fig. 13 is obtained where e_0^0 and e_1^0 stand for *normal* and *derailment*, e_0^1 and e_1^1 for $fast \cup medium \cup slow$, and e_2^1 and e_3^1 for *touch* and *separate*, respectively.

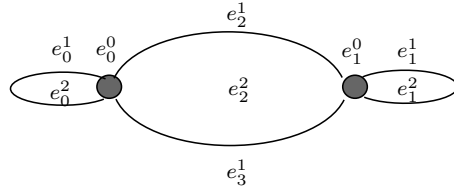


Fig. 13 State Transition Diagram on Cellular Space Level

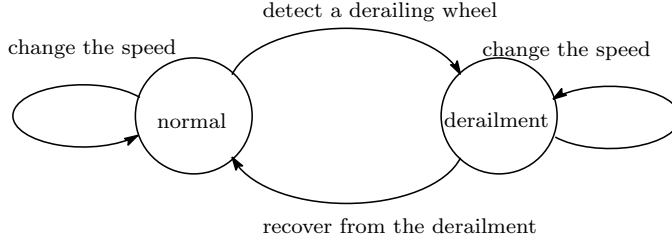


Fig. 14 State Transition Diagram at View Level

3. The presentation level

For each transition, the corresponding event is identified by giving its explanation as in Fig. 14.

4.1.2 Constructing $Y \times I$

The product space of agents and interval $Y \times I$ is constructed from Y and I as follows.

$$\begin{aligned} Y &= \{brain, sensors, wheel_controllers, wheels\}, \\ I &= \text{Any sequence generated by events,} \\ events &= \{touch, separate, fast, medium, slow\}. \end{aligned}$$

Therefore, $I = (touch|separate|fast|medium|slow)^*$. From this definition, $H : Y \times I \rightarrow B$ is easily obtained.

1. The cellular space level

Using B and $Y \times I$, E is obtained as follows. As E represents the π -calculus processes for each agent, the agent process diagram for each agent is obtained by $p^{-1}(B)$. Each event is specified by the sequence of actions to explain how each agent behaves. The following is an example. When the right_sensor detects touching the black line, $touch$ consists of the following actions.

- $touch_1$: the right_sensor sends detection of touching the black line to the right_wheel_controller,
- $touch_2$: the right_wheel_controller receives it
- $touch_3$: the right_wheel_controller sends message of stopping the wheel to the right_wheel, and
- $touch_4$: the right_wheel receives it.

Then, $\hat{H} : Y \times I \rightarrow E$ is defined as follows.

$$\hat{H} : (y, i) \in Y \times I \rightarrow e_i^{a,1} \in E \text{ such that}$$

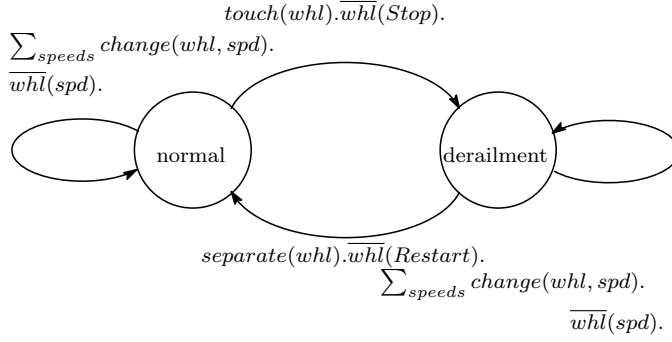


Fig. 15 Agent Process Diagram for *wheel_controllers*

$\hat{H}(normal, touch_1) \rightarrow e_2^{s,1}$ where $e_2^{s,1}$ stands for $\overline{touch}(wheel)$,
 $\hat{H}(normal, touch_2) \rightarrow e_2^{wc,1}$ where $e_2^{wc,1}$ stands for $touch(wheel)$,
 $\hat{H}(normal, touch_3) \rightarrow e_2^{wc,1}$ where $e_2^{wc,1}$ stands for $\overline{wheel}(Stop)$, and
 $\hat{H}(normal, touch_4) \rightarrow e_2^{w,1}$ where $e_2^{w,1}$ stands for $wheel(Stop)$.

Other events are defined in a similar way.

2. The presentation level

π -calculus processes is obtained for each agent using homotopy \hat{H} where each action is transformed into π -calculus. For *wheel_controller*, two processes are defined according to *behavios*. P_{wc}^{normal} is for *normal* and $P_{wc}^{derailment}$ for *derailment*. P_{wc}^{normal} receives two kind of events. One is from *brain* to change speed and the other from *sensors* to inform of touching the black line. These are represented as follows.

$$\begin{aligned}
 & P_{wc}^{normal}(controller, change, touch, separate, \\
 & \hspace{15em} speeds) \\
 = & \sum_{speeds} change(whl, spd).\overline{whl}(spd).P_{wc}^{normal}(\dots) \\
 & + touch(whl).\overline{whl}(Stop).P_{wc}^{derailment}(\dots).
 \end{aligned}$$

In the same way, $P_{wc}^{derailment}$ is described by the following process.

$$\begin{aligned}
 & P_{wc}^{derailment}(controller, change, touch, separate, \\
 & \hspace{15em} speeds) \\
 = & \sum_{speeds} change(whl, spd).\overline{whl}(spd).P_{wc}^{derailment}(\dots) \\
 & + separate(whl).\overline{whl}(Restart).P_{wc}^{normal}(\dots).
 \end{aligned}$$

The agent process diagram for *wheel_controllers* are depicted in Fig. 15.

The agent process diagram for *wheels* is minimized to a single state by applying state minimization. The process is described by the following equation.

$$P_{wd}(driver) = whl(msg).P_{wd}(\dots).$$

In the same way, processes for *sensors* are obtained by the following equations.

$$\begin{aligned}
P_s^{normal}(sensor, touch, separate, wheel) \\
&= \overline{touch}(wheel).P_s^{derailment}(\dots). \\
P_s^{derailment}(sensor, touch, separate, wheel) \\
&= \overline{separate}(wheel).P_s^{normal}(\dots).
\end{aligned}$$

The process for *brain* is obtained by the following equation.

$$\begin{aligned}
P_b(change1, change2, wheel1, wheel2, speeds) \\
&= \sum_{speeds} \overline{change1}(wheel1, spd).P_c(\dots) \\
&\quad + \sum_{speeds} \overline{change2}(wheel2, spd).P_c(\dots).
\end{aligned}$$

Finally, all instances corresponding to each agent are defined and executed in parallel.

$$\begin{aligned}
&CarRobot \\
&= (New\ ControllerA, ControllerB, \\
&\quad DriverA, DriverB, SensorA, SensorB, \\
&\quad ChangeA, ChangeB, WheelA, WheelB, \\
&\quad TouchA, TouchB, SeparateA, SeparateB, \\
&\quad Speeds = \{Slow, Medium, Fast\}) \\
&\quad P_{wc}^{normal}(ControllerA, ChangeA, TouchA, \\
&\quad SeparateA, Speeds) \\
&\quad | P_{wc}^{normal}(ControllerB, ChangeB, TouchB, \\
&\quad SeparateB, Speeds) \\
&\quad | P_{wd}(DriverA) | P_{wd}(DriverB) \\
&\quad | P_s^{normal}(SensorA, TouchA, SeparateA, \\
&\quad WheelA) \\
&\quad | P_s^{normal}(SensorB, TouchB, SeparateB, \\
&\quad WheelB) \\
&\quad | P_b(ChangeA, ChangeB, WheelA, WheelB, \\
&\quad Speeds).
\end{aligned}$$

4.2 Using the Homotopy Extension Property

XMOS XC programs is created for Car Robot. XMOS is featured by non von Neumann architecture. The architecture combines a number of processing cores, each with its own memory and I/O systems, on a single chip. The processing core, called Xcore, is an event driven multi thread processing system. XMOS XC is a C like programming language. Using the HEP in Fig 2, Car Robot is constituted by X as the π -calculus

processes for the agents, A as the π -calculus processes for an agent, and Y^I as the programs. These spaces are obtained as follows.

4.2.1 Constructing Y^I

$\hat{K} : X \rightarrow Y^I$ is constructed where X is the space of the π -calculus processes for the agents, Y is the space of the programs for the agents, I is any sequence of events, and Y^I is the space of path functions. Therefore, \hat{K} maps $x \in X$ to $\lambda \in Y^I$ and $\lambda : I \rightarrow Y$. As the set of events is defined by $\{touch, separate, fast, medium, slow\}$, any sequence of events is defined by $(touch|separate|fast|medium|slow)^*$ and λ is explained as execution of a series of programs along the sequence of events. Therefore, program piece $Y^{t \in I}$ is implemented so that it satisfies some $x \in X$ where $\hat{K}(x) = Y^t$.

For example, the program related to agent *sensors* is implemented. Assuming that A is the π -calculus processes of *sensors*, the program is obtained by $K : A \rightarrow Y^I$. For event *touch*, $K(P_s^{normal}(\dots) = \overline{touch}(wheel).P_s^{derailment}(\dots)) = Y^{touch}$. Program piece Y^{touch} is implemented in the following program codes where an event is achieved by an input port, an output port, a timer or a channel. In this program, the black line is detected by "light when pinseq(...) :> void" statement. If the lack line is detected, this information is sent through channel *touch*.

```
while (1) {
  if (state == normal) {
    light when pinseq(detect_derailing):> void;
    touch <: 1;
    state = derailing;
  }
}
```

In the same way, program piece $Y^{separate}$ is implemented using $K(P_s^{derailment}(\dots) = \overline{separate}(wheel).P_s^{normal}(\dots)) = Y^{separate}$ for event *separate*. As no transitions are defined for *fast*, *medium* and *slow*, program pieces Y^{fast} , Y^{medium} and Y^{slow} are empty. Then, implementation of the program for agent *sensors* is completed as follows,

```
void sensor(in port light, chanend touch,
  chanend separate) {
  int state;
  timer tmr;
  unsigned time;
  state = normal;
  while (1) {
    if (state == normal) {
      light when pinseq(detect_derailing):> void;
      touch <: 1;
      state = derailing;
    } else if (state == derailing) {
      light when pinseq(detect_normal):> void;
      separate <: 1;
      state = normal;
    }
    tmr :> time;
    time += delay;
    tmr when timerafter(time):>void;
  }
}
```

The program is implemented for agent *wheel_controllers*. Its π -calculus processes are represented using sum + such as

$$P_{wc}^{normal}(\cdot) = \sum_{speeds} change(\dots).\overline{whl}(\cdot).P_{wc}^{normal}(\cdot) + touch(\cdot).\overline{whl}(\cdot).P_{wc}^{derailment}(\dots).$$

Sum + is realized using select statement. Guards such as *change* and *touch* are realized by channels. The program framework for sum + is shown below.

```
select {
  case touch :> cmd:
  ...
  break;
  case change :> cmd:
  ...
  break;
}
```

When *A* is of *wheel_controllers*, then, the entire program for agent *wheel_controllers* is obtained as follows,

```
void wheel_controller(chanend touch,
  chanend separate,
  chanend wheel, chanend change) {
  int state;
  int cmd;
  state = normal;
  while (1) {
    if (state == normal) {
      select {
        case touch :> cmd:
          wheel <: stop;
          state = derailing;
          break;
        case change :> cmd:
          wheel <: cmd;
          break;
      }
    } else if (state == derailing) {
      select {
        case separate :> cmd:
          wheel <: restart;
          state = normal;
          break;
        case change :> cmd:
          wheel <: cmd;
          break;
      }
    }
  }
}
```

If *A* is of *wheels*, then, the program for agent *wheels* is implemented as follows where the speed received by channel *wheel* and taking *stop, restart, fast, medium* or *slow* as its value are realized by switch statement,

```
void wheel_driver(out port pulse,
  chanend wheel) {
  int cmd, pulse_state = 1, skip = 0;
  unsigned current_on_duty = on_duty,
  current_off_duty = off_duty,
  fast_on_duty = current_on_duty * 2;
  medium_on_duty = current_on_duty;
```

```

slow_on_duty = current_on_duty / 2;
fast_off_duty = current_off_duty;
medium_off_duty = current_off_duty;
slow_off_duty = current_off_duty;
unsigned pt;
timer ptmr;
ptmr :> pt;
pt += current_on_duty;
while (1) {
  select {
    case ptmr when timerafter(pt) :> int:
      if(pulse_state) pt += current_on_duty;
      else pt += current_off_duty;
      pulse_state = !pulse_state;
      if(!skip) pulse <: pulse_state;
      break;
    case wheel :> cmd:
      switch(cmd) {
        case stop:
          skip = 1;
          break;
        case restart:
          skip = 0;
          break;
        case slow:
          current_on_duty = slow_on_duty;
          current_off_duty = slow_off_duty;
          break;
        case medium:
          current_on_duty = medium_on_duty;
          current_off_duty = medium_off_duty;
          break;
        case fast:
          current_on_duty = fast_on_duty;
          current_off_duty = fast_off_duty;
          break;
      }
      break;
  }
}
}
}

```

Then, the instances of each agent as the main program where parallel processing is realized by par statement,

```

int main(void) {
  chan changeA, changeB, touchA, touchB,
  separateA, separateB, wheelA, wheelB;
  par {
    commander(changeA, changeB);
    sensor(left_light, touchA, separateA);
    sensor(right_light, touchB, separateB);
    wheel_controller(touchA, separateA,
                    wheelA, changeA);
    wheel_controller(touchB, separateB,
                    wheelB, changeB);
    wheel_driver(left_wheel, wheelA);
    wheel_driver(right_wheel, wheelB);
  }
  return 0;
}

```

The XMOS XC program for Car Robot has the same topological structure as the π -calculus processes for Car Robot. As the invariants defined in the π -calculus processes are kept in the program, it is not necessary to verify the topological structure of the program. It is only necessary to verify whether the processes are transformed correctly to the program. However, important structures of π -calculus processes such as sum $+$, parallel $|$, and guards consisting of event inputs and outputs are automatically generated without faults. Like Pict[Pierce and Turner(1997)], it is not hard to implement a compiler from the π -calculus to XMOS XC.

5 Conclusions

Two important properties of homotopy theory, the HLP and the HEP, have been used to designing and modeling the enterprise system and the real-time embedded system in a bottom-up way and a top-down way, respectively. The HLP and the HEP are useful for composing and decomposing a system in designing and modeling it. These properties are used as a tool for climbing-up or climbing-down the component hierarchy. On the other hand, the IMAH is used as another tool to go through the abstraction hierarchy. The component hierarchy and the abstraction hierarchy, which are important concepts in designing and modeling cyberworlds, can be handled in a formal way using the modern mathematics. Since the invariants defined at the abstraction levels of IMAH are inherited at all the lower levels, our approach can avoid much of verification and testing.

References

- [Havey(2005)] Havey M (2005) Essential Business Process Modeling. O'Reilly Media, Inc, Cambridge
- [Hennessy(2001)] Hennessy M (2001) A Distributed Pi-Calculus. Cambridge University Press, Cambridge
- [Kunii(2005)] Kunii TL (2005) Cyberworlds -theory, design and potetial-. The Transactions of The Institute of Electronics, Information and Communication Engineers E88-D(5):790–800
- [Kunii and Ohmori(2006)] Kunii TL, Ohmori K (2006) Cyberworlds: Architecture and modeling by an incrementally modular abstraction hierarchy. The Visual Computer 22(12):949–964
- [May(2007)] May D (2007) Communicating process architecture for multicores. The 30th Communicating Process Architectures Conference pp 21–32
- [Milner(1999)] Milner R (1999) Communicating And Mobile Systems: Pi-Calculus. Cambridge University Press, Cambridge
- [Ohmori and Kunii(2006)] Ohmori K, Kunii TL (2006) An incrementally modular abstraction hierarchy for linear software development methodology. Int Conf on Cyberworlds 2006 pp 216–223
- [Ohmori and Kunii(2007a)] Ohmori K, Kunii TL (2007a) Development of an accounting system. ICEIS2007 pp 437–444
- [Ohmori and Kunii(2007b)] Ohmori K, Kunii TL (2007b) The mathematical structure of cyberworlds. Int Conf on Cyberworlds 2007 pp 100–107
- [Ohmori and Kunii(2008a)] Ohmori K, Kunii TL (2008a) Mathematical modeling of ubiquitous systems. Int Conf on Cyberworlds 2008 pp 69–74
- [Ohmori and Kunii(2008b)] Ohmori K, Kunii TL (2008b) A pi-calculus modeling method for cyberworlds systems using the duality between a fibration and a cofibration. Int Conf on Cyberworlds 2008 pp 363–370
- [Ohmori and Kunii(2009)] Ohmori K, Kunii TL (2009) Mathematical foundation for designing and modeling cybeworlds. Int Conf on Cyberworlds 2009 pp 80–87

-
- [Pierce and Turner(1997)] Pierce BC, Turner DN (1997) Pict a programming language based on the pi-calculus. Indiana University CSCI Technical Report 476 pp 1–26
- [Sangiorgi and Walker(1999)] Sangiorgi D, Walker D (1999) The Pi-Calculus: A Theory of Mobile Processes. Cambridge University Press, Cambridge
- [Sieradski(1992)] Sieradski AJ (1992) An introduction to topology and homotopy. PWS-Kent Publishing Company, Boston
- [Spanier(1966)] Spanier EH (1966) Algebraic topology. Springer-Verlag, New York