Distributed Cooperative Control Architectures for Automated Manufacturing Systems

by O. Patrick Kreidl

S.B. in Electrical Engineering, George Mason University, 1994

Submitted to the Department of Electrical Engineering & Computer Science in Partial Fulfillment of the Requirements for the Degree of

Master of Science

at the Massachusetts Institute of Technology

February 1996

Author .....
Department of Electrical Engineering & Computer Science
February 5, 1996

Certified by ....
M. Athans
Professor of Electrical Engineering & Computer Science
Thesis Supervisor

Certified by .....
J. N. Tsitsiklis
Professor of Electrical Engineering & Computer Science
Thesis Supervisor

Accepted by .....
F. R. Morgenthaler
Professor of Electrical Engineering & Computer Science
Chairman, Departmental Committee for Graduate Students

Distributed Cooperative Control Architectures for Automated Manufacturing Systems

by O. Patrick Kreidl

Submitted to the Department of Electrical Engineering & Computer Science, Massachusetts Institute of Technology, on February 5, 1996 in Partial Fulfillment of the Requirements for the Degree of Master of Science.

Abstract

Technological advances in the areas of micro-computing, communications, and system science are leading the way for new capabilities in automated manufacturing systems. Emerging control area networks, consisting of intelligent components that can share information over a communication link without relying on a central controller, are motivating increasingly complex, large-scale, information-intensive automation for the factory-floor. In response to these trends, coupled with a rising interest for greater efficiency and flexibility, we describe *distributed cooperative control*, or the notion of intelligent sub-systems coordinating their local decisions over a communication network to achieve aggregate system goals. A distributed system topology is presented, conceptually encompassing many advanced automation capabilities for a restricted class of discrete-part processes, and forms the basis for an overall system model which logically supports a variety of system-level distributed cooperative control architectures. In order to systematically characterize alternative architectures, we propose an architectural modeling procedure derived from object-oriented principles. It is argued that a resulting architectural model captures, in addition to the flow and processing of discrete parts, a precise abstraction of the decision control strategy, communication capabilities, and information management scheme corresponding to the candidate architecture. For the purposes of scheduling and control, the modeling procedure not only allows for *qualitative* comparisons of alternative architectures but, by direct consequence of the object-oriented paradigm, also supports *quantitative* experimental evaluation through simulation.

Thesis Supervisor: M. Athans
Title: Professor of Electrical Engineering & Computer Science

Thesis Supervisor: J. N. Tsitsiklis
Title: Professor of Electrical Engineering & Computer Science

Acknowledgments

I foremost would like to thank my immediate thesis supervisors, Professor Michael Athans and Professor John Tsitsiklis, for their direct guidance and support. The many interesting discussions with them and Professor Dimitri Bertsekas in the early stages of this research will be treasured always. I would also like to thank Christopher Peters for his research collaboration, general friendship, helpful insight into the problem, and especially for extending the presented ideas into a successful simulation.

I am very fortunate to work in the Laboratory for Information and Decision Systems at MIT, which provides a stimulating research environment and surrounds me with many wonderful people. In particular I wish to thank Wesley McDermott, Steve Patek, Joel Douglas, and Leonard Lublin for their generally helpful suggestions and valuable friendship.

I would like to thank my parents (Wigbert and Adolfine), my older sisters (Astrid and Sandrine), and the rest of my family and friends for their continuous love and support throughout all of my endeavors. I also wish to acknowledge several of my past supervisors, namely Professor Gerald Cook and Professor Bijan Jabbari of George Mason University, and Dr. Gary Trusty of the Naval Research Laboratory, all of whom have given me valuable advice and significantly influenced my career objectives.

Finally, I would like to acknowledge the support of our industrial sponsor, Square-D Company/Groupe Schneider, and am especially grateful to Dennis Brandl for his trust and guidance in this research. The frequent interactions with our sponsor proved helpful for refining the ideas presented in this work.

Contents

Chapter 1

Introduction

1.1 Background

The growing complexity of current industrial manufacturing systems is motivated by the requirements for global competition in terms of greater flexibility, higher efficiency, better product quality, and lower overall costs. Meanwhile, technological advances in the areas of communications, system science, and computer hardware/software allow for the potential implementation of advanced manufacturing technology. The integration of knowledge from many disciplines is necessary to achieve this advanced level of manufacturing automation, and calls for a more unified understanding of potential modern manufacturing systems.

The above ideas form the underlying theme of Computer-Integrated Manufacturing (CIM) [1], [2], [3]. CIM strives to integrate all stages in a product life-cycle, including taking orders from customers, production planning, product design, process scheduling, etc. into a facility-wide information system while minimizing human intervention. Additional manufacturing philosophies which have received much attention in recent decades are Flexible Manufacturing Systems (FMSs) [1] and Just in Time (JIT) production management [4, pages 1–5].

FMSs are characterized by collections of *flexible machines* that are tied together by an automated material handling system, usually all under computer control. A typical FMS may include flexible machines with part buffers, inspection stations, and a set of pallets (upon which parts are loaded) that can travel along an automated guideway. Each flexible machine is equipped with tools to perform a variety of tasks, where an associated setup time exists between task change-overs. In most cases, FMSs are capable of producing more than one part type, known as *product flexibility*. Also, by providing redundancy of tasks among different machines, a part process may have multiple routes within the FMS, known as *process flexibility*.

The JIT production management philosophy is an overall manufacturing approach with the straightforward goal to produce, at an acceptable quality, required parts by the time at which they are required. One realization of this philosophy is known as Kanban which achieves the JIT goal at the expense of large work-in-progress inventories. With the implementation of FMSs featuring process flexibility, an alternative

realization might be termed *manufacturing-to-order*, where the focus is on filling customer orders by manufacturing required parts as these orders are received rather than maintaining large inventories. The JIT paradigm also introduces the notion of *product families*, or the grouping of products which share common manufacturing attributes (e.g., geometry, weight, required materials, etc.), and promotes the notion of a *product-based factory layout.*

1.2 Motivation

The evolution towards more and more sophisticated levels of factory automation began with the introduction of the Programmable Logic Controller, or PLC, in the early 1970s. The development of the PLC was a first step towards addressing the needs of industrial automation with programmed solutions versus previous wired combinational/sequential logic solutions [5]. These wired implementations suffered from their size (weight and volume), lack of flexibility with regard to subsequent development, complexity of fault-finding and repairs, and the inability to meet the continuing demand to address more complex problems. It was quickly realized among the manufacturing community that PLC implementations provided significant improvements over these shortcomings and were better suited to handle the overall needs of industrial automation.

Today, the industrial community is again facing a similar challenge. The primary role of the PLC is as an instrument of command, taking control of a process or a machine. However, in response to today's demanding industrial needs, PLCs have also been used (beyond their intended design limits) to perform additional functions including diagnostic duties, routing decisions in multi-stream configurations, numerical and signal processing, data management functions, and even communications. Although the capabilities of PLCs have been greatly enhanced, these PLC-based implementations are more suitable for time-critical command, data collection, and monitoring rather than for intelligent control. It is also becoming evident that the purely PLC-based factory-floor systems, when applied to today's complex line configurations, suffer from lack of flexibility (in terms of required re-programming with regard to line augmentation), lack of robustness to failures, and ad-hoc design procedures.

The current trend in the implementation of factory-floor control systems is to replace the PLC-based systems, consisting of point-to-point wiring between a controller and its components, with *control area networks*, consisting of intelligent components that can share information over a communication link and coordinate decisions without relying on a central controller [6], [7], [8]. By eliminating the miles of wiring required in traditional PLC-based implementations, immediate benefits include easier installation, higher reliability, and lower maintenance costs. Thus, efforts in the industrial research community are being focused on the development and specification of standards, such as Fieldbus [9], for these device-level networks. Furthermore, control area networks yield the potential for computer-based control and distributed machine control, based on intelligent sensors and actuators, as well as a host of other bene-

fits such as system health monitoring, adaptive response, and software-configurable manufacturing using intuitive programming languages.

In complex manufacturing systems, such as in *multi-product/multi-stream* cases, many local control area networks may be required. If these individual networks are designed separately, with no communication or coordination between them, the resultant system might be termed the *union* of "automated islands" rather than the *integration* of "automated islands." Therefore, the definition of a global architecture for today's complex manufacturing systems requires, at an early stage, the analysis of the interdependencies linking local sub-systems within the aggregate system. A complete study of modern factory-floor control systems not only requires an understanding of device-level coordination (within a sub-system between local PLCs, sensors, actuators, etc.) but also an understanding of system-level coordination (between the sub-systems that run the machine stations, transport system, etc.).

Moving from PLC-based implementations to modern computer network-based implementations also demands a greater design emphasis on the supporting software requirements. In other words, to fully realize the additional capabilities available through the use of computers in factory-floor control, it is important that these software implementations are not merely "soft-wired" versions of "hard-wired" automation. Because of the inherently distributed aspects of today's complex factory-floor control systems, approaches for developing modular software for manufacturing applications are gaining momentum [4], [10], [11]. Furthermore, driven by the need for system flexibility, the supporting software should allow for high-level re-programming and be compatible with *open systems* (as termed in recent computer science literature). Open systems have the characteristics of concurrency, asynchrony, decentralized distributed control based on possibly inconsistent information, and high dynamic adaptability requirements resulting from unrestricted joining and leaving sub-systems [12]. These characteristics are naturally becoming part of the modern manufacturing environment, where both information gathering and decision making can be logically and physically distributed among sub-systems, and desired robustness to failures and line augmentation requires a system with high dynamic adaptability.

1.3 Thesis Objectives

This thesis surveys the current state of research and development for computer control of flexible manufacturing systems suggesting a growing interest in large-scale, information-intensive automation. In response to this trend we describe *distributed cooperative control*, or the notion of intelligent sub-systems coordinating their local decisions over a communication network to achieve aggregate system goals. A distributed system topology is presented, conceptually encompassing many advanced automation capabilities for a restricted class of discrete-part processes, and forms the basis for an overall system model that logically supports a variety of system-level distributed cooperative control architectures. In order to systematically characterize alternative architectures, we propose an architectural modeling procedure derived from object-oriented principles. It will be argued that a resulting architectural model

captures, in addition to the flow and processing of discrete parts, a precise abstraction of the decision control strategy, communication capabilities, and information management scheme corresponding to the candidate architecture. For the purposes of scheduling and control, the modeling procedure will not only allow for *qualitative* comparisons of alternative architectures but also support *quantitative* experimental evaluation through simulation.

1.4 Organization of Thesis

In Chapter 2, we motivate the application of distributed cooperative control for manufacturing and highlight the features that characterize system-level architectures. Chapter 3 introduces a conceptual system model and describes the assumptions on the class of manufacturing processes it represents. Chapter 4 discusses several principles and techniques used in early modeling stages of object oriented system development and, based on these ideas, identifies a systematic procedure for characterizing distributed cooperative control architectures. In Chapter 5, we demonstrate the procedure in an example by obtaining a precise abstraction of system component requirements under two simple candidate architectures. Chapter 6 gives a brief summary and concludes with suggestions for further research.

Chapter 2

Distributed Cooperative Control

The recent advances in high performance computers and high-speed communication networks have brought about a new emphasis among the industrial community regarding the analysis, design, and control of automated manufacturing systems. Control area networks, heterarchical information management strategies, modeling and analysis through simulation, and object-based software systems are among the many recently proposed concepts reflecting an increasing interest in large-scale, flexible, information-intensive manufacturing automation. The appropriate integration of these concepts and support from advanced technology is a promising direction towards meeting the demanding requirements of complex modern factory-floor control systems.

The distributed cooperative view of a control system recognizes that very complex systems, as for modern manufacturing, are beyond direct centralized control [12]. There are many choices to be made regarding the organization of a distributed cooperative system. In our manufacturing context, we view a factory-floor under distributed cooperative control at two extreme levels. The *device-level* consists of the highly time-critical control of an arrangement of PLCs, sensors, actuators, and other basic industrial automation devices comprising a *sub-system.* Standards for device-level configurations are emerging [8], [9].

At the *system-level*, where we view each sub-system as a single unit, the control system is collectively determined by:

1. the arrangement of the aggregate system into self-contained, intelligent sub-systems,

2. the local decision rules and information present within the sub-systems, and

3. the required inter-communication between the networked sub-systems which allows them to coordinate their local decisions.

In this thesis, we propose a systematic modeling procedure that captures the system-level features of candidate distributed cooperative control architectures. To be most useful, a modeling procedure should be supported by analysis techniques for verifying that, under the given architecture, the system is guaranteed to perform correctly and according to a certain level of performance. In our manufacturing context, *system*

correctness refers to the guarantee that the distributed components operate and coordinate in a manner such that parts are delivered reliably and correctly built. *System performance* refers to the guarantee that system correctness is carried out efficiently, meeting demand deadlines within the constraints posed by the architecture.

2.1 System-level Architectures

At the system-level, a distributed cooperative control architecture can be broadly characterized by three interdependent features: the decision control strategy, communication capabilities, and information management scheme. Each concept is in itself an entire field of active research. We merely present a brief overview of each area and suggest a few related references.

2.1.1 Decision Control Strategy

In an environment where sub-systems are physically and logically distributed, a fundamental concern is that the system operate correctly and according to a desired level of performance in the presence of limited and possibly inconsistent information. A general treatment of numerical methods for distributed (and parallel) computing systems, as well as an overview of generic issues concerning distributed (and parallel) computation, can be found in [13]. To further complicate matters, the real-world manufacturing environment is subject to stochastic effects. Dynamic programming [14] is a central algorithmic method that provides a unifying framework for sequential decision-making under uncertainty. The method centers around identifying a control policy, or strategy, such that a certain cost, or mathematical expression of what is considered an undesirable outcome, is minimized. Only recently, with emerging simulation-based approximation techniques, has the practical application of dynamic programming to large-scale problems been possible [15]. In the absence of a central control mechanism, a first issue is the breakdown of the "total" control strategy into "smaller" strategies assigned to different cooperating sub-systems.

2.1.2 Communication Capabilities

In our context, each sub-system consists of a number of basic automation devices connected by a control area network. In modern large-scale systems, the local decisions within each sub-system must be appropriately coordinated to achieve a set of desired system-level goals. Therefore, we assume a hierarchical network structure consisting of many control area networks interconnected by a reliable backbone. Continuing advances in optical technology promise to make optical networks, with desirable qualities of large bandwidth and low-loss transmission, ideal backbones that reliably support high-volume traffic [16]. Given such a reliable network, it remains to identify the "who talks to whom" relationship (and the types of messages exchanged) among sub-systems enabling a decision control strategy and information management scheme to be properly implemented.

2.1.3 Information Management Scheme

Successful coordination of the decisions within a set of cooperating sub-systems depends on identifying efficient information management schemes. Traditional hierarchical strategies [17], [18] are useful for systems with a "master-slave" type of relationship among components. This type of relationship is only a fraction of the types present among distributed cooperative system components. As an alternative, heterarchical strategies [19], [20] assume an environment where autonomous sub-systems (with isolated information and control) communicate with one another as peers rather than through a restrictive hierarchical chain. An ultimate goal for heterarchical strategies is to minimize the system's required global information, meanwhile simplifying the interaction between sub-systems and avoiding the possibilities of excessive loading on communications.

2.2 Modeling and Analysis of Architectures

According to [21], a modeling procedure should be expressive, mapping all the properties of interest in the manufacturing system into the model. A model should be easy to use and understand and should allow analysis and decision-making concerning the real-world environment. Particular to scheduling and control of modern network-based manufacturing systems, a model should emphasize (in contrast to traditional transfer line models that emphasize only the flow and processing of material [22]) details of the decision control strategy, communication capabilities, and information management scheme as well as the flow and processing of material. The authors classify modern manufacturing systems as *reactive* and identify several challenges associated with modeling these types of systems. For example, there is no simple mathematical transformation that can represent the system and the communication among system components is hard to represent. Also, components are concurrently and asynchronously active and the variety of states that the system can have is typically too large to enumerate.

A concise survey on modeling approaches for manufacturing systems is given in [23]. Of these approaches, the author argues that modeling and evaluation through simulation is currently the *only* viable approach to a detailed analysis, as needed for scheduling and control purposes, of modern manufacturing systems. Simulation can handle considerable complexity and can be made to mimic the real system as accurately as time (and patience) permit. The main drawbacks are that construction of a simulation is a formidable task and evaluation, consisting of possibly many simulation runs, can become computationally expensive.

2.2.1 Software Support

At the system level, a software model should capture the relevant aspects of a candidate control architecture, including the organization of the distributed system into logical modules and the specification of the coordination and necessary communication between modules. Once modeled in sufficient detail, design changes can be

simulated and the corresponding system correctness and performance can be evaluated. With regard to eventual implementation, development of software compatible with open systems (where distributed components are linked by a communication network) promises, as argued in Chapter 1, improvements in system robustness to manufacturing failures and line augmentation. Finally, following a unified software approach, fusing modeling, simulation, and implementation into one overall design procedure, yields the potential for introducing special purpose tools for analysis, design, and control of automated manufacturing lines.

2.2.2 Object-Oriented Approaches

An object-oriented approach to developing software is based on modeling real-world objects and their interactions [24], [25]. The approach focuses on the initial software modeling of concepts inherent to the particular application and views the actual software implementation as merely a final translation of the model into a programming language. This approach naturally provides the desired "common thread" from initial software modeling through final implementation.

The fundamental attributes of the object-oriented paradigm seem well-suited for achieving the software modeling, simulation and implementation goals mentioned above. In particular, the encapsulation of data and methods into self-contained objects and the interaction between objects via the exchange of messages correlate well with the idea of modular sub-systems which coordinate their local decisions via network communication. Furthermore, data/method encapsulation coupled with inter-object communication exploiting polymorphic behavior provide a foundation for developing the desired open systems environment. The concept of inheritance, a powerful abstraction that allows object classes to share similarities while preserving their differences, is a useful way to broadcast change through the software system and potentially minimizes the re-programming required for line augmentation.

Objects in the modeling world are designed to match their physical counterparts in the real world. As a result, it seems reasonable to develop factory-floor control system design and implementation tools based on object-oriented simulations. We envision developing a "true-to-life" object-oriented simulation which captures the device-level control signals as well as the system-level control strategies. Once the simulation is shown to perform adequately (i.e., the correctness of the decision rules is verified and the simulated performance is satisfactory), we merely replace the simulated device-level control signals with physical connections to the corresponding devices of the actual control system. This potential one-to-one correspondence between object-oriented simulation and object-oriented implementation would minimize the need for "ad-hoc" procedures involved in current design techniques. Many examples in recent literature provide evidence that object-oriented approaches are indeed finding their place in design and implementation of factory-floor control systems [10], [26], [27], [28], [29], [30], [31].

Chapter 3

Description of System Model

In this chapter, we formulate a conceptual framework which aims to capture the typical features and complexity associated with control of likely future automated manufacturing systems. The framework is built upon a general system model, abstracting a distributed cooperative control manufacturing environment to include only the fundamental *system-level* components. In other words, the internal processes of each distributed component are completely handled by some unspecified device-level configuration. While we now state several restrictions regarding the representable class of manufacturing processes, the system model will be shown to encompass many features of advanced and sophisticated manufacturing automation.

We consider only discrete-part manufacturing processes where part types travel through a collection of flexible processing stations via an automated, pallet-based, transfer mechanism.[1] Perhaps the two most severe restrictions are that the system model, as described in this chapter, does not allow assembly or cycling operations. Assembly operations arise in cases where two intermediate part types need to be joined into a single part type before further processing continues. Cycling arises when a part type may need to return to a workstation for repeated processing.[2]

Under the above restrictions, we devote this chapter to describe and parameterize a generic system model that encompasses many advanced manufacturing concepts and automation capabilities. Abstractly, the model involves ideas of product and process flexibility, flexible workstations with automatic part inspection capabilities, and buffers with possibly unrestricted queue disciplines ("Totestackers"). In a manufacturing environment where such advanced capabilities are assumed available, with proper specification of the distributed cooperative control strategies, a manufacturing-to-order capability seems especially feasible.

---

[1] Fortunately, a significant portion of modern manufacturing applications satisfy these properties.

[2] Assembly and cycling operations are common in modern manufacturing processes, especially in the semiconductor industry. Extending the system model to handle these operations is marked as a topic for further research.

Distributed Components	
${}^kL$	kth layer controller in system
I_i	ith input station of system
O_i	ith output station of system
${}^kM_i$	ith flexible machine within kth layer
${}^kB_i$	ith buffer within kth layer
${}^kJ_i$	ith guideway junction within kth layer
${}^kG_i$	ith "horizontal" guideway link within kth layer
${}^kH_i$	ith "vertical" guideway link within kth layer

Topology Parameters	
n	number of machine layers in system
m_k	number of stations in kth layer
${}^kT_i$	set of tasks performed by ith station within kth layer
j_k	number of guideway junctions in kth layer
g_k	number of "horizontal" guideway links in kth layer
h_k	number of "vertical" guideway links in kth layer

Table 3.1: Notation for Figure 3-1

3.1 General Topology

We begin development of our system model by describing the general distributed system topology depicted in Figure 3-1, which we term the *MPMS (Multi-Product/Multi-Stream) Topology*. Each notated component (see Table 3.1) conceptually represents an individual device-level arrangement. Thus, from this point-of-view, the topology merely serves as a logical representation of how system-level components might typically be arranged in a distributed cooperative manufacturing environment. We emphasize that another goal of this model is one of generality – to provide a mapping, into a reference system-level form, of any manufacturing system topology in the restricted class under consideration.

3.1.1 Factors Influencing Topology

Notion of Layers

A significant feature inherent to the proposed topology is the separation of the distributed components into logical *layers* (these layers are separated by the dashed vertical lines in Figure 3-1). The notion of layers stems from placing several restric-

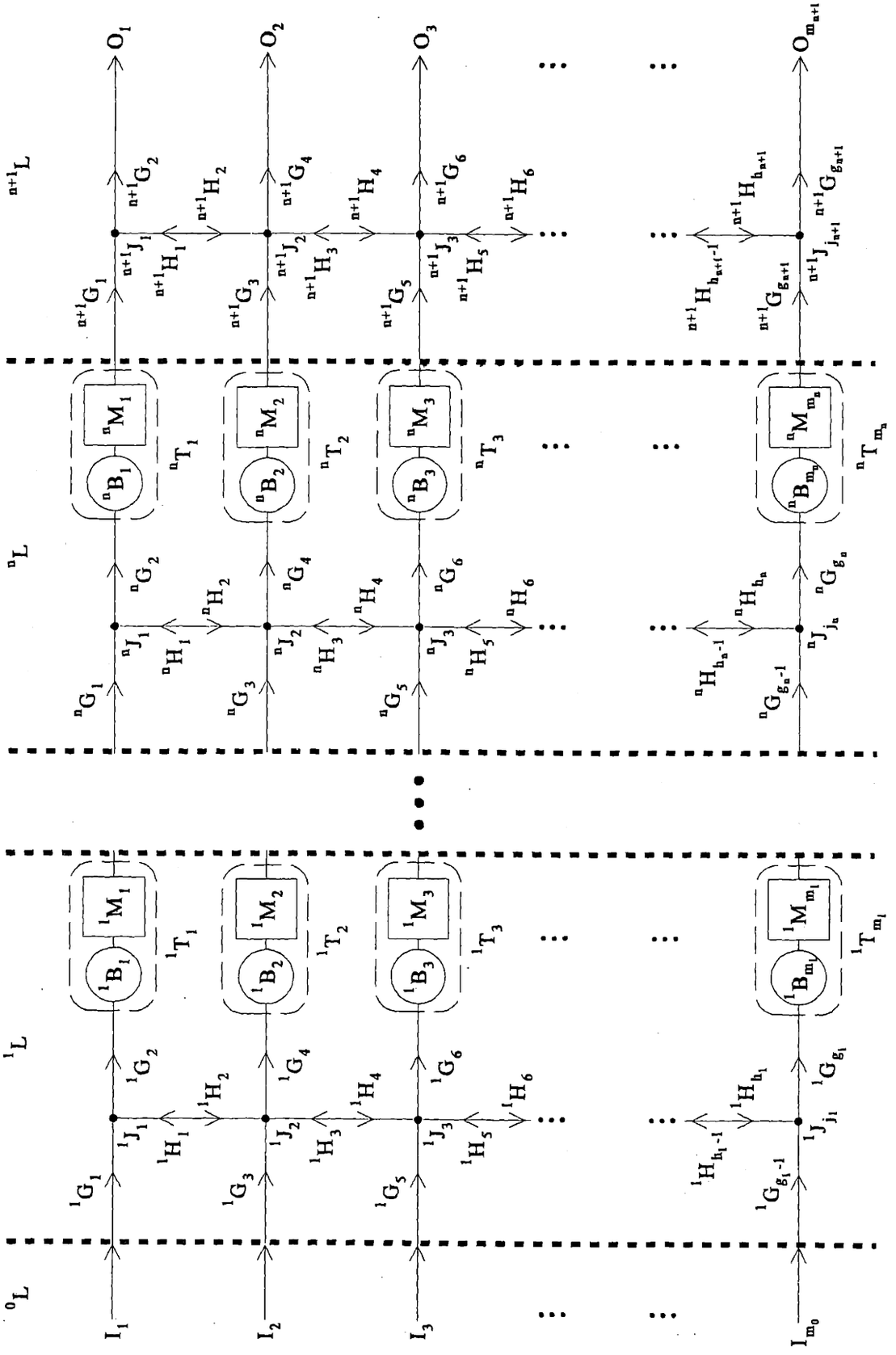


Figure 3-1: General MPMS Topology

tive assumptions regarding the flow of parts through the system. First and foremost, we require that a pallet undergoes one and only one task in each layer. Each raw pallet is assumed to be prepared in the input layer at one of several input stations, and via the corresponding input stream, enters the first machine layer. Upon entering each machine layer, a pallet is routed to a *single* workstation, undergoes a *single* required task, and moves into the next layer. After completion of its final task, the pallet is routed to its appropriate output station in the output layer.

Product-Based Layout

The requirement for a pallet-based transfer mechanism naturally implies that any instance of the topology be associated with a single part family (i.e., set of part types with similar size, weight, and geometry). At our level of abstraction, we uniquely distinguish a part type within a part family by its associated task sequence (i.e., ordered sequence of required tasks). Thus, to handle the notion of a part family while maintaining the part flow assumptions, we must ensure by layered arrangement of flexible machines that the first required task of every part type in the part family be done in the first layer, the second required task of every part type be done in the second layer, and so on. In summary, the necessary presence of a station in a particular layer is dependent on the tasks that the station is capable of performing and at what stage within the part type sequences these tasks fall.

The "Bypass" Operation

In order for the topology to satisfy the layered, product-based assumptions as stated above, it would be necessary to impose a very restrictive requirement on a representative part family. Namely, each part type within the part family must have the same number of tasks in its sequence as the number of layers in the corresponding topology. To avoid this we allow a part type, if its task sequence requires, to "bypass" operation in a layer by some specified mechanism.[3] Now for the topology to satisfy the layered, product-based assumptions, the number of tasks in the *longest* task sequence of a part family determines the number of layers. In addition, for any task sequences one task shorter we add one bypass operation into the sequence. For task sequences two tasks shorter we add two bypass operations, and so on.

Remarks

Introducing this layered, product-based structure into our system model provides several advantages. First, the flow control aspect of the problem is separated into layer-by-layer stages where each subsequent stage of the manufacturing process corresponds to what occurs in each subsequent layer. Second, there results an inherent

[3]One such mechanism might be to have an unrestricted queue discipline in a workstation. Then, a part type intending to bypass a layer may arrive to the workstation's buffer, enter the machine when next available, forego an operation, and exit the workstation into the next layer.

pattern in the set of system events possible in each stage. More specifically, it follows that each given part type enters a layer with a known *required task*. If machine resources in a layer allow for task redundancy (i.e., where the same task can be performed at more than one workstation), then a choice exists for a pallet's *assigned workstation*. This choice may be performed in a dynamic fashion. At each stage, any current information gained through a given cooperative strategy is incorporated into the decision process.[4]

In order to hedge against the possibility of persistent backlog between layers, it is important to appropriately "balance" resources in each layer. For instance, if all tasks of one layer are much shorter than tasks in the following layer, then additional machine resources and task redundancy in the slower layer may be appropriate. The *balancing of layers* is a fundamental performance issue of the MPMS system model.[5]

3.1.2 Manufacturing-to-Order Concept

Notion of Priority

We incorporate the manufacturing-to-order concept into the model by assuming that the system receives customer orders, each with an associated deadline, by some higher-level manager. An order consists of a specified composition of part types and an associated *order priority* measures its deadline requirements relative to those of other existing orders. Entering pallets, carrying parts corresponding to a particular order, are tagged with a relative *pallet priority* within the order. For example, pallets carrying part types that require longer processing may have a higher pallet priority than other part types within the same customer order. For the idea of priority to be most effective, it is important to assume some mechanism for which a high priority pallet may be hurried through the layers and avoid waiting behind low priority pallets. One such mechanism might be unrestricted queue disciplines in station buffers.

We also assume that each workstation may have the capability to inspect a part on an outgoing pallet and determine whether its task has been adequately completed. If the inspection results in a part failure, then the system must ensure that the faulty part is exited from the line and a replacement part is introduced at the input (probably with a higher pallet priority to timely fulfill the order). As a final detail, we require all finished parts pertaining to a unique customer order to be collected at the same output station. Here one may, for example, envision the capability of automatically packaging completed orders for shipping.

---

[4]From a software system point-of-view, recognizing and exploiting these "design patterns" can lead to object-based software that is more reusable [32]. In our context, one might argue that this implies a system more readily adaptive to line augmentation and failures.

[5]Layer balancing, scheduling, efficient routing, and other performance issues related to distributed cooperative control and the MPMS system model are beyond the scope of this thesis. These issues are marked as topics for further research and the reader is referred to [33] for recent work related to this area.

Remarks

Striving for generality, one might investigate the notion of orders as a way for describing many different types of demand on a manufacturing system. For instance, a traditional part-throughput system may be represented as many same priority orders consisting of a single part type. A batch-throughput system may be represented as orders consisting of many of the same part type. We may view building inventory as simply introducing into the system very low priority orders consisting of the desired part types. The notion of orders, order priority, and pallet priority inevitably impact the system-level decision strategies in a complex way. For example, the performance merit changes from minimizing the "part cycle time," or the average time to fulfill a part request, to minimizing the "order cycle time," or the average time to complete the final part request of an order.

3.2 Parameterization of System Model

In this section, we introduce some terminology/notation to help with concisely abstracting the relationship between the MPMS topology and several of the concepts described above. The parameterization is developed within a general setting and two instructional examples follow which illustrate its application to more specific cases.

3.2.1 Terminology/Notation

MPMS Topology

For any sample of the general topology shown in Figure 3-1, we identify the associated parameters (see Table 3.1) with the following notation:

`MPMS Topology:`

n = number of machine layers

m_k = number of stations in kth layer; $k = 0, 1, \ldots, n+1$

${}^kT_i$ = set of tasks performed by ith station in kth layer; $i = 1, 2, \ldots, m_k$; $k = 0, 1, \ldots, n+1$

j_k = number of guideway junctions in kth layer; $k = 1, 2, \ldots, n+1$

g_k = number of "horizontal" guideway links within kth layer; $k = 1, 2, \ldots, n+1$

h_k = number of "vertical" guideway links within kth layer; $k = 1, 2, \ldots, n+1$

Station Family

A `station family` refers to the set of input stations, workstations, and output stations (characterized by the set of tasks they perform) available to include in a sample topology. For example, the expression

$$ {}^{k}T_i \in \begin{cases} \text{task set } 1 \\ \text{task set } 2 \\ \quad \vdots \\ \text{task set } j \end{cases} \quad (j \text{ types}); \quad \text{all } i; \quad k = 1, 2, \ldots, n $$

denotes that the task set of the ith station to be chosen for the kth layer is one of the j types listed. Including the input and output layers, we use the following notation:

`Station Family:` $n =$ number of layers

$$ {}^{0}T_i \in \begin{cases} \text{input task set } 1 \\ \text{input task set } 2 \\ \quad \vdots \\ \text{input task set } j \end{cases} \quad (j \text{ types}); \quad \text{all } i $$

$$ {}^{k}T_i \in \begin{cases} \text{task set } 1 \\ \text{task set } 2 \\ \quad \vdots \\ \text{task set } j \end{cases} \quad (j \text{ types}); \quad \text{all } i; \quad k = 1, 2, \ldots, n $$

$$ {}^{n+1}T_i \in \begin{cases} \text{output task set } 1 \\ \text{output task set } 2 \\ \quad \vdots \\ \text{output task set } j \end{cases} \quad (j \text{ types}); \quad \text{all } i $$

When given a sample `MPMS topology` the `station family` directly follows. It is only when a topology is not initially known, but rather needs to be determined, that being given or assuming a `station family` becomes essential.

Layer Task Support

Given an `MPMS topology` and thus ${}^{k}T_i$ for all $i = 1, 2, \ldots, m_k$ and $k = 0, 1, \ldots, n+1$ are known, we define the `layer task support` as

`Layer Task Support:` $n =$ number of layers

$$ {}^{k}L \leftarrow \bigcup_{i=1}^{m_k} {}^{k}T_i; \quad k = 0, 1, \ldots, n+1 $$

This notation denotes the obvious statement that for the MPMS topology, the set of tasks available in layer k is the union of the task sets of all stations within layer k. In other words, the layer task support is no more than the layered breakdown of tasks available in a given arrangement of stations.

Part Family

We define a part family to be comprised of p part types where a part type x, for $x \in \{\text{part type names}\}$, is characterized by its associated task sequence Ts_x. For each part type, a task sequence is composed of an input task, an ordered set of n machine tasks (possibly including bypass operations), and an output task. More precisely, denoting ${}^kT_X$ as the task required in Layer k for part type X, we notate a part family by

$$\texttt{Part Family:} \quad p = \text{number of part types}; \quad x \in \{\text{part type names}\}$$

$$Pt_x \;\rightarrow\; Ts_x = ({}^0T_x, \underbrace{{}^1T_x, {}^2T_x, \ldots, {}^nT_x}_{n \text{ machine tasks}}, {}^{n+1}T_x); \qquad \text{all } x$$

Layer Task Requirements

Given a part family and thus ${}^kT_x$ for all $x \in \{\text{part type names}\}$ and $k = 0, 1, \ldots, n+1$, we define the layer task requirements as

$$\texttt{Layer Task Requirements:} \quad n = \text{number of layers}; \quad x \in \{\text{part type names}\}$$

$${}^kL \;\rightarrow\; \bigcup_{\text{all } x} \{{}^kT_x\}; \qquad k = 0,\, 1,\, 2,\, \ldots,\, n,\, n+1$$

This notation denotes the statement that for any part family, the set of tasks required in each layer k has as its elements the kth machine task in the task sequence of every part type in the part family. In other words, the layer task requirements are no more than the layered breakdown of tasks required to realize a given part family.

3.2.2 Product-Based Layout of Topology

Using the above notation/terminology, we now express the product-based layout relationship inherent in the system model. The relationship is a result of the logical layered arrangement of the MPMS topology, the assumptions regarding the flow of part types through the layers, and the concept of part families whose part types are characterized by associated task sequences. We make the straightforward observation that, for an MPMS topology to realize a part family, it is necessary for the layer

`task requirements` to be *identical to* or *a subset of* the `layer task support`.

$$\bigcup_{\text{all } x} \{{}^kT_x\} \subseteq \bigcup_{i=1}^{m_k} {}^kT_i; \quad k = 0, 1, \ldots, n+1 \tag{3.1}$$

where
n = number of machine layers
$x \in$ {part type names}
${}^kT_x$ = task required in kth layer by part type x
m_k = number of stations in kth layer
${}^kT_i$ = set of tasks performed by ith station in kth layer

In other words, if each layer k of the `MPMS topology` has at least a single workstation capable of performing the kth machine task of every part type's task sequence, then equation (3.1) holds and we say the `part family` is realizable by the topology.

3.2.3 Examples

We now illustrate the use of the system model parameterization described above. In the first example, we are given an `MPMS topology` and asked to determine a sample `part family`. This requires obtaining the `layer task support` resulting from the given topology and then choosing a `part family` whose `layer task requirements` satisfy equation (3.1). Similarly, the second example gives a `part family` and `station family` and asks to determine a a sample `MPMS topology`. This case requires obtaining the `layer task requirements` resulting from the given `part family` and then choosing a `topology` (within the constraints of the `station family`) whose `layer task support` satisfies equation (3.1).

Example 1

Problem Statement: Considering the given topology shown in Figure 3-2, determine a sample `part family`.

Solution: From the figure, we readily obtain the following `topology` and `station family` parameterization:

`MPMS Topology:`

$$n = 2$$

$$m_k = 2; \quad k = 0, 1, 2, 3$$
$${}^0T_i = U; \quad i = 1, 2$$
$${}^1T_i = \{T_1, T_2\}; \quad i = 1, 2$$
$${}^2T_i = \{T_3, T_4\}; \quad i = 1, 2$$
$${}^3T_i = U; \quad i = 1, 2$$

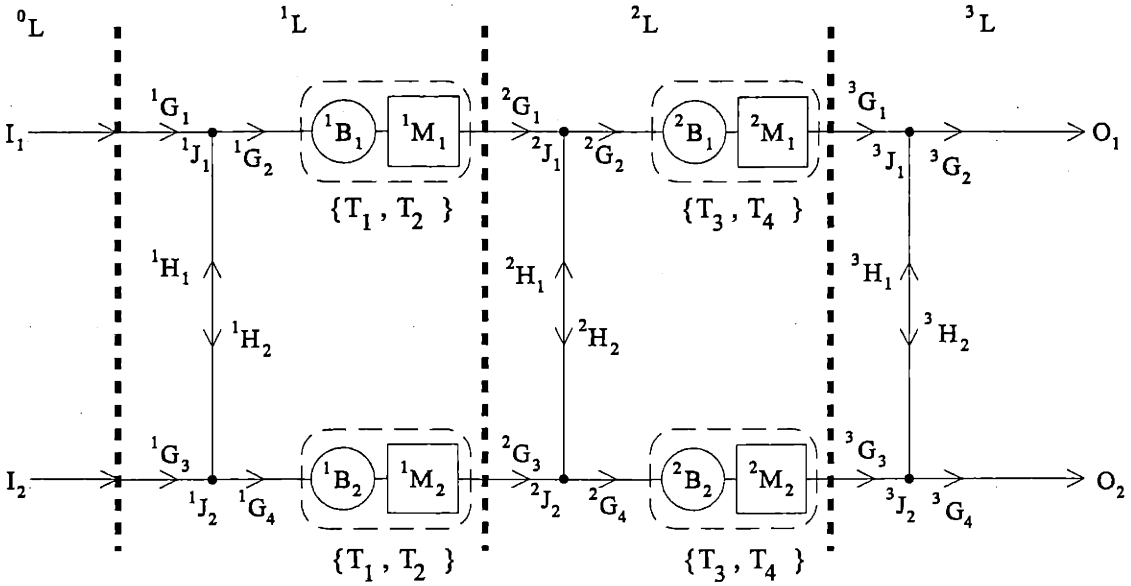


Figure 3-2: Given MPMS Topology for Example 1

$$
\begin{aligned}
j_k &= 2; \quad k = 1, 2, 3 \\
g_k &= 4; \quad k = 1, 2, 3 \\
h_k &= 2; \quad k = 1, 2, 3
\end{aligned}
$$

`Station Family:` $n = 2$

$$
\begin{aligned}
{}^0T_i &\in \text{universal set } U \quad \text{(any type)}; \quad \text{all } i \\
{}^kT_i &\in \begin{cases} \{T_1, T_2\} \\ \{T_3, T_4\} \end{cases} \quad \text{(2 types)}; \quad \text{all } i; \quad k = 1, 2 \\
{}^3T_i &\in \text{universal set } U \quad \text{(any type)}; \quad \text{all } i
\end{aligned}
$$

Note that we assume input (output) stations are available that can perform the required input (output) tasks for any `part family` to be determined. Thus, until we determine the sample `part family`, we ignore the `task requirements` and `task support` of the input and output layers. The `layer task support` corresponding to the given `topology` is

`Layer Task Support:` $n = 2$

$$
\begin{aligned}
{}^0L &\leftarrow U \\
{}^1L &\leftarrow \{T_1, T_2\} \\
{}^2L &\leftarrow \{T_3, T_4\}
\end{aligned}
$$

$$^3L \quad \leftarrow \quad U$$

By equation (3.1), an appropriate `part family` is any collection of part types, all with 2-layer task sequences, which realize `layer task requirements` satisfying the following conditions:

`Layer Task Requirements:` $x \in U, \quad n = 3$

$$\begin{aligned} {}^0L &\rightarrow \bigcup_{\text{all } x} \{{}^0T_i\} \subseteq U \\ {}^1L &\rightarrow \bigcup_{\text{all } x} \{{}^1T_i\} \subseteq \{T_1, T_2\} \\ {}^2L &\rightarrow \bigcup_{\text{all } x} \{{}^2T_i\} \subseteq \{T_3, T_4\} \\ {}^3L &\rightarrow \bigcup_{\text{all } x} \{{}^3T_i\} \subseteq \{T_3, T_4\} \end{aligned}$$

It can be shown that the following sample `part family` satisfies these relations with equality:

`Part Family:` $p = 3; \quad x \in \{A, B, C\}$

$$\begin{aligned} Pt_A &\rightarrow Ts_A = (T_{A_i}, T_1, T_3, T_{A_o}) \\ Pt_B &\rightarrow Ts_B = (T_{B_i}, T_1, T_4, T_{B_o}) \\ Pt_C &\rightarrow Ts_C = (T_{C_i}, T_2, T_4, T_{C_o}) \end{aligned}$$

Example 2

Problem Statement: Given the following `part family` and `station family`, determine a sample `MPMS topology`.

`Part Family:` $p = 3; \quad x \in \{A, B, C\}$

$$\begin{aligned} Pt_A &\rightarrow Ts_A = (T_{A_i}, T_1, T_2, T_3, T_{A_o}) \\ Pt_B &\rightarrow Ts_B = (T_{B_i}, T_1, T_3, T_{B_o}) \\ Pt_C &\rightarrow Ts_C = (T_{C_i}, T_1, T_4, T_1, T_{C_o}) \end{aligned}$$

`Station Family:` $n =$ number of layers

$$^0T_i \quad \in \quad \{T_{A_i}, T_{B_i}, T_{C_i}\} \quad \text{(one type)}; \quad \text{all } i$$

$$
{}^{k}M_i \quad \in \quad {}^{k}T_i \begin{cases} \{T_1\} \\ \{T_3\} \\ \{T_1,\, T_3\} \\ \{T_2,\, T_4\} \end{cases} \qquad \text{(four types)}\,; \quad \text{all } i\,; \quad k = 1, 2, \ldots, n
$$

$$
{}^{n+1}T_i \quad \in \quad \{T_{A_o}, T_{B_o}, T_{C_o}\} \quad \text{(one type)}\,; \quad \text{all } i
$$

Solution: Notice for this `part family` that $n = 3$ and the task sequence for part type B contains one less task than for the other part types. Thus, Ts_B is required to include an additional task, a "bypass" operation. The `layer task requirements` follow from the given `part family`, and due to the required bypass operation, there are three cases.

`Layer Task Requirements:`

$$
{}^{0}L \;\rightarrow\; \{T_{A_i}, T_{B_i}, T_{C_i}\}
$$

Case 1: Pt_B bypasses ${}^{1}L$, $\quad Ts_B = (T_{B_i}, -, T_1, T_3, T_{B_o})$

$$
\begin{aligned}
{}^{1}L &\rightarrow \{T_1\} \\
{}^{2}L &\rightarrow \{T_1, T_2, T_4\} \\
{}^{3}L &\rightarrow \{T_1, T_3\}
\end{aligned}
$$

Case 2: Pt_B bypasses ${}^{2}L$, $\quad Ts_B = (T_{B_i}, T_1, -, T_3, T_{B_o})$

$$
\begin{aligned}
{}^{1}L &\rightarrow \{T_1\} \\
{}^{2}L &\rightarrow \{T_2, T_4\} \\
{}^{3}L &\rightarrow \{T_1, T_3\}
\end{aligned}
$$

Case 3: Pt_B bypasses ${}^{3}L$, $\quad Ts_B = (T_{B_i}, T_1, T_3, -, T_{B_o})$

$$
\begin{aligned}
{}^{1}L &\rightarrow \{T_1\} \\
{}^{2}L &\rightarrow \{T_2, T_3, T_4\} \\
{}^{3}L &\rightarrow \{T_1, T_3\} \\
{}^{4}L &\rightarrow \{T_{A_0}, T_{B_o}, T_{C_o}\}
\end{aligned}
$$

For case 2, by equation (3.1) an appropriate `topology` is any 3-layer arrangement of `station family` members which realizes a `layer task support` satisfying the following conditions:

Layer Task Support: $n = 3$

$$
\begin{aligned}
{}^0L &\leftarrow \bigcup_{i=1}^{m_0} \{{}^0T_i\} \supseteq \{T_{A_i}, T_{B_i}, T_{C_i}\} \\
{}^1L &\leftarrow \bigcup_{i=1}^{m_1} \{{}^1T_i\} \supseteq \{T_1\} \\
{}^2L &\leftarrow \bigcup_{i=1}^{m_2} \{{}^2T_i\} \supseteq \{T_2, T_4\} \\
{}^3L &\leftarrow \bigcup_{i=1}^{m_3} \{{}^3T_i\} \supseteq \{T_1, T_3\} \\
{}^4L &\leftarrow \bigcup_{i=1}^{m_4} \{{}^4T_i\} \supseteq \{T_{A_o}, T_{B_o}, T_{C_o}\}
\end{aligned}
$$

It can be shown that the sample topology in Figure 3-3 satisfies these relations with equality. The corresponding notation reads

MPMS Topology:

$$
\begin{aligned}
n &= 3 \\
(m_0, m_1, m_2, m_3, m_4) &= (2, 2, 2, 3, 2) \\
{}^0T_i &= \{T_{A_i}, T_{B_i}, T_{C_i}\}; \quad i = 1, 2 \\
{}^1T_i &= \{T_1\}; \quad i = 1, 2 \\
{}^2T_i &= \{T_2, T_4\}; \quad i = 1, 2 \\
({}^3T_1, {}^3T_2, {}^3T_3) &= (\{T_1\}, \{T_3\}, \{T_1, T_3\}) \\
{}^4T_i &= \{T_{A_o}, T_{B_o}, T_{C_o}\}; \quad i = 1, 2 \\
(j_1, j_2, j_3, j_4) &= (2, 2, 2, 2) \\
(g_1, g_2, g_3, g_4) &= (4, 4, 5, 5) \\
(h_1, h_2, h_3, h_4) &= (2, 2, 2, 2)
\end{aligned}
$$

Similar topologies can be determined for cases 1 and 3 and, as an interesting aside, we note the following. Suppose we have a sample topology with layer task support being the union of all three cases, namely

Layer Task Support: $n = 3$

$$
\begin{aligned}
{}^0L &\leftarrow \{T_{A_i}, T_{B_i}, T_{C_i}\} \\
{}^1L &\leftarrow \{T_1\} \\
{}^2L &\leftarrow \{T_1, T_2, T_3, T_4\}
\end{aligned}
$$

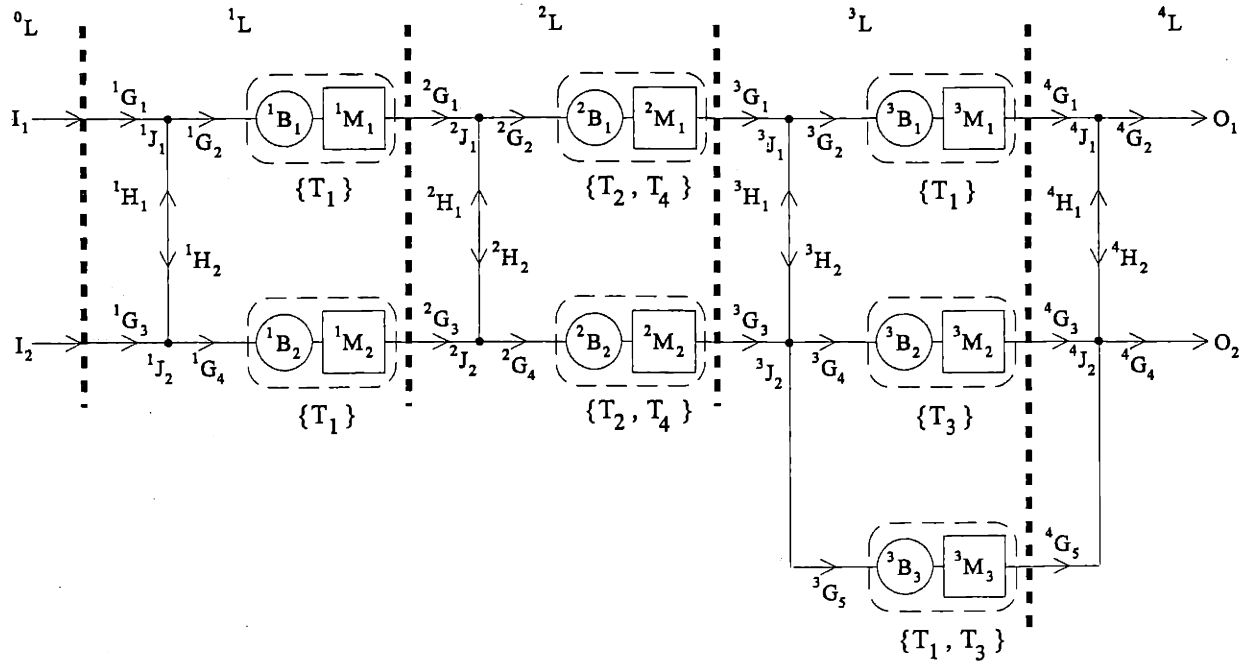


Figure 3-3: Sample MPMS Topology for Example 2, Case 2

$$
\begin{aligned}
{}^3L &\leftarrow \{T_1, T_3\} \\
{}^4L &\leftarrow \{T_{A_0}, T_{B_o}, T_{C_o}\}
\end{aligned}
$$

Then, depending on the location of the bypass operation, part type B could be identified by three different task sequences. This is a small example of an additional freedom that the bypass operation allows us to introduce in the model. Suppose we have a `part family` whose *longest* sequence has size m. If we choose the number of layers $n > m$, we can arrange the `topology` (`station family` permitting) in such a way that for each part type there exists, in addition to the choice of which station to visit in each layer, the choice of which layer(s) to bypass. One might argue that allowing this additional freedom has the potential for extending the class of manufacturing topologies and part families representable by the system model.

3.3 Chapter Summary

Presented in this chapter is a system model and a concise notation for its parameterization. We acknowledge that this model formulation is abstract but we attempt to provide a general conceptual framework that captures the typical features and complexity associated with likely future automated manufacturing systems. To this effect, we incorporated the ideas of product flexibility, process flexibility, flexible workstations, just-in-time production management by way of the manufacturing-to-order concept and, most importantly, the notion of distributed intelligent sub-systems cooperating via a reliable communication network. The system model logically supports a variety of system-level distributed cooperative control architectures.

Chapter 4

On Describing Architectures

4.1 Preview

In the context of this thesis, a distributed cooperative control architecture is primarily characterized by three system-level features:

1. a decision control strategy, or set of high-level decision rules between and within system components,

2. a supporting reliable communication network, or a specified "who talks to whom" relationship among system components, and

3. a particular information management scheme, or choice of relevant data, its interpretation, and methods for its exchange among components.

There are an infinite number of choices to be made when specifying a candidate architecture. Thus, it is essential to identify a systematic procedure for determining and representing architectures. When faced with a complicated system development problem, such as specifying distributed cooperative control architectures, it is important to preserve the freedom to make system development decisions as long as possible by avoiding premature commitments to details. For this reason, it is useful to view a system development problem at different "appropriate" levels of abstraction.

In this chapter we propose an approach, guided by object-oriented modeling principles and techniques, for capturing at a *system-level* abstraction much of the relevant information pertaining to the system correctness of a candidate architecture.[1] Assuming a manufacturing environment conceptually compatible with the system model presented in Chapter 3, we identify an architectural modeling procedure which relies on scenario-based object analysis ideas.

[1] In a first-iteration analysis, while even system correctness requirements are not fully specified, it would be premature to also include system performance requirements.

4.2 Object Analysis Model

The object-oriented design methodology [24],[25] is a systematic approach for developing complex software systems. The procedure emphasizes first capturing the real-world concepts and interactions inherent to the software application and then merely translating them into a final software implementation. It views the software development process in a common way from early stages of modeling and design through final stages of implementation.

In the early stages, the analysis phase of the overall methodology is concerned with building a correct and understandable model of the real-world situation and its important properties. Starting from a (usually incomplete) statement of the problem, the resulting analysis model is a precise, concise abstraction of *what* a system must do, avoiding any implementation decisions and without restricting *how* implementation is to be done, and captures three essential aspects of the system: the objects and their relationships (object model), the dynamic flow of control (dynamic model), and the functional transformation of data subject to constraints (functional model). The three sub-models are orthogonal but cross-linked parts of the description for a complete analysis model. In short, the functional model specifies "what happens," the dynamic model specifies "when it happens," and the object model specifies "what entities are involved in what happens."

4.2.1 Description of Sub-Models

Object Model

The object model is the most fundamental part of an analysis model, as it is necessary to specify *what* is changing before specifying *when* or *how* it changes. It describes the static structure of the objects in a system and their relationships by way of object diagrams. An object diagram is a graph whose nodes are object classes, labeled with data attributes and methods particular to each class, and whose arcs represent associations, or specific relationships among classes.

Dynamic Model

The dynamic model describes the interactions among objects, how they change over time, and is also significant in specifying the control aspects of the system. It contains multiple state diagrams, one state diagram for each class with important dynamic behavior. A state diagram is a graph whose nodes are object states and whose directed arcs represent transitions between states caused by events. In complex systems which exhibit properties of hierarchy, concurrency, and communication, the formalism of statecharts [34] is useful for drawing structured state diagrams with nested contours to show complex dynamic relationships between objects.

Functional Model

The functional model describes the computations within a system. It shows how output data values are derived from input data values without regard for the order in which the values are computed. It is built up of multiple data-flow diagrams, each representing a computation as data flows from external inputs, through operations and internal data stores, to external outputs. A data-flow diagram is a graph whose nodes can be processes that transform data, actor objects that produce or consume data, and data store objects that store data passively. Data-flows are represented by directed arcs labeled with the data parameter being passed.

4.2.2 Scenario-Based Analysis

A particularly challenging aspect of developing object analysis models is initially defining the objects and their associated requirements and behavior. Scenario-driven analysis principles and techniques provide a structured approach for capturing an object analysis model. The idea is based on formulating typical system scenarios, or thought experiments, at different "appropriate" levels of abstraction and deriving a set of related objects from each scenario. Taking the union of these derived sets yields a first-iteration composite object analysis model.

There are many useful tools and constructs available that aid in the effort of formulating a set of system scenarios. One such tool is a *script table* and is merely a format for organizing the logical flow of events that characterize a scenario. A single row of the table corresponds to a single event within the scenario and typically involves the exchange of messages between an initiator object and a number (possibly zero) of participating objects.

4.3 Scenario-Based System Model

With the analogy depicted in Figure 4-1, we motivate using object modeling concepts as a guide for modeling distributed cooperative control architectures. Specifically, the procedure we propose relies on applying scenario-based object analysis ideas to capture a precise, first-iteration, system-level specification of a candidate architecture. The set of scenarios required for our procedure roughly span three levels of abstraction: the system scenario, the action scenarios, and the system events.

4.3.1 Organization of Scenario Abstraction

The *system scenario* rests at the top of our "abstraction hierarchy" and features a conceptual framework that represents a presumed manufacturing environment. As we will see shortly, the first step of the proposed architectural modeling procedure is to formulate a (usually incomplete) description of a candidate architecture within the framework of the system scenario. For this reason a system scenario should strive for generality, supporting a variety of decision control strategies, communication capabilities, and information management schemes.

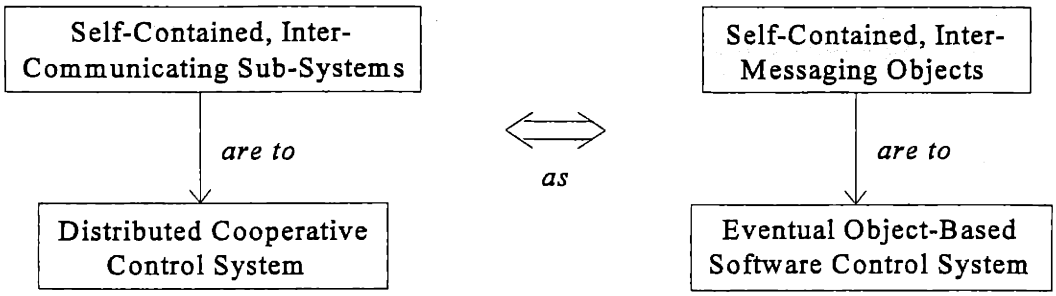


Figure 4-1: Analogy Motivating Architectural Modeling Procedure

An event-driven evolution of a system scenario is an abstraction of its temporal behavior, substituting the notion of time with series of events. In our context, each basic element of the scenario evolution begins in response to a previous event and terminates by initiating a subsequent event, hence continuing the evolution. These basic elements are termed *system events* and comprise the lowest-level of our abstraction hierarchy. Typically, a system event involves an interaction between an initiating system component and a number (possibly zero) of participating system components.

The *action scenarios* serve as an intermediate unit of packaging between the system scenario and the basic building blocks of system events. Action scenarios are logical constructs, grouping a subset of system events to capture one particular stage of the system evolution. In determining a choice of action scenarios, it is desirable to partition them by system events which are inherent to the given system scenario (i.e., events that occur under any candidate architecture). Adhering to this precaution arguably leads to improved consistency over a set of architectural models specified within a system scenario.

The relationship between our three levels of abstraction can be summarized as follows: each *action scenario* represents a logical set of architecture-dependent *system events* likely to occur between a pair of architecture-independent *system events* inherent to the *system scenario*. Any successful application of the proposed architectural modeling procedure rests with appropriate formulation of these three levels of abstraction. Namely, we require the system scenario and action scenarios to be defined such that a candidate architecture is completely characterized by the choice of system events.

4.3.2 General MPMS System Scenario

This section presents a general system scenario, based on the MPMS system model, upon whose instances we apply the proposed architectural modeling procedure. As discussed in Chapter 3, the general MPMS topology is a conceptual representation of a set of flexible machines, arranged in layers according to a restricted product-based layout, operating within a distributed cooperative manufacturing environment. Exploiting the structure and organization incorporated into the MPMS model, we now describe the system scenario in two complementary parts: a modular description and an appropriate choice of the set of action scenarios.

The modular description of the MPMS system scenario identifies the individual system components and the static relationships between them. The description is given in the form of an object diagram. The action scenarios characterize the logical evolution of the general system scenario, and are expressed in terms of a state diagram. We emphasize that, for both parts, we only concern ourselves with capturing relationships inherent to the system scenario (i.e, relationships present regardless of the choice of architecture).

Modular Description

The modular description of the system scenario is given in Figures 4-2, 4-3, and 4-4 in the form of an object diagram organized into three sheets. Sheet 1 captures the product-based layout relations present in the general MPMS topology, Sheet 2 describes how the system components are statically interconnected, and Sheet 3 describes the component relationships established to accommodate the manufacturing-to-order concept (note we are not yet concerned with identifying the data and methods inside the components as this depends on the choice of architecture). Several of the components appear on multiple sheets. We indicate other sheets that refer to a component with numbers inside circles contiguous to the class box. A detailed description of each sheet is given below.

Sheet 1: Product-Based Layout Relations

We start with a `Part Family` which maps to many `MPMS Topologies`. The `Part Family` is made up of p `Part Types`, each having an associated manufacturing process consisting of a unique `Task Sequence`. A `Task Sequence` is made up of an `Input Task`, an ordered set of n machine `Tasks`, and an `Output Task`. A corresponding `MPMS Topology` is made up of an `Input Layer`, an ordered set of n machine `Layers`, and an `Output Layer`. There is a tight relationship between the `Task` composition of `Task Sequences` in the `Part Family` and the `Workstation` composition of the `Layers` in the `MPMS System`. Namely, each kth `Layer` must contain at least one `Workstation` that performs the kth `Task` of the `Task Sequence` for every `Part Type` in the `Part Family`.

The `Input Layer`, viewed as the 0th layer of the `MPMS Topology`, is made up of m_0 `Input Stations`, each of which is composed of an `Input Buffer` and `Input Stream`. The `Input Stream` in the ith `Input Station` can perform an `Input Task Set` ${}^0T_i$. Each `Layer` can be conceptually divided into a `Transport Sub-layer` and a `Machine Sub-layer`. In the kth `Layer`, the `Transport Sub-layer` is made up of g_k horizontal guideway `Links`, h_k vertical guideway `Links`, and j_k guideway `Junctions`. The `Machine Sub-layer` of the kth `Layer` has in it m_k `Workstations`, each made up of a `Buffer` and a `Machine`. The `Machine` in the ith `Workstation` of the kth `Layer` can perform the machine `Task Set` ${}^kT_i$. Finally, the `Output Layer`, viewed as the $(n+1)$th layer of the `MPMS System`, is structured the same as a `Layer` except that its `Machine Sub-layer` consists of m_{n+1} `Output Stations` (rather than a set of `Workstations`). Each `Output Station` is composed of an `Output Buffer` and an `Output Stream`. The `Output Stream` in the ith `Output Station` can perform an `Output Task Set` ${}^{n+1}T_i$.

Sheet 2: System Component Relations

A given Link will have two Endpoint Components – an Upstream Component and the other a Downstream Component. Because we assume unidirectional links, a Link is always an incoming Link to its Downstream Component and an outgoing Link to its Upstream Component, as shown by the derived associations.

A Junction is the only possible Upstream Component that is an element of the same Layer as the given Link. When the Upstream Component is in the previous Layer, it is a Workstation unless the previous Layer is the Input Layer, in which case it is an Input Station. Given the Upstream Component is an Input Station, then the component has no additional incoming nor outgoing Links. If the Upstream Component is a Workstation, then it has exactly one other incoming Link that is a member of the same Layer as the Workstation. Finally, if the Upstream Component is a Junction, then it may have zero or more additional outgoing Links and one or more other incoming Links, all members of the same Layer as the Junction.

Similarly, a Junction or a Workstation are the only Downstream Components which can be members of the same Layer as the given Link. If the Link is in the Output Layer, then the Downstream Component is also in the Output Layer and it is either an Output Station or a Junction. Given the Downstream Component is an Output Station, then it has no additional incoming nor outgoing Links. When in an interim Layer and the Downstream Component is a Workstation, then it has exactly one other outgoing Link that is a member of the next Layer. Finally, if the Downstream Component is a Junction, then it may have zero or more additional incoming Links and one or more other outgoing Links, all of which are members of the same Layer as the Junction.

Sheet 3: Manufacturing-to-Order Relations

When an Order, consisting of one or more Part Types, is introduced into the system, the Output Stream at which the completed Part Types will be collected must be determined. Each Part Type of the Order is introduced into the system at one of m_0 Input Stations where a corresponding Pallet is appropriately prepared and tagged with ID information.

It is assumed, in general, that a Pallet carries with it two essential pieces of information. The first is its current destination which, when the Pallet is in the Input Layer, is composed of an Input Task and the Input Station that will perform it. When the Pallet is in a machine Layer, the current destination is composed of a Task and a Workstation that will perform it. When in the Output Layer, the Pallet's current destination is composed of an Output Task and an Output Station that will perform it. The destination task of a Pallet is uniquely determined by the Task Sequence corresponding to the Pallet's Part Type, and in which Layer the Pallet is currently located. The destination station is assigned in a manner that depends on the specifics of a control strategy and the (estimated) state of the system. Regardless of exactly how the Pallet's destination station is assigned, the result is assumed to be recorded on the Pallet itself. Note that in the Output Layer, the destination station is the Output Station assigned to collect the Order that the

particular Pallet belongs to.

The other essential piece of information recorded on the Pallet is its current location. The location, at the Layer level, is either the Input Layer, interim Layer, or Output Layer. When in the Input Layer, the Pallet must be in a particular Input Station either waiting in the Input Buffer or undergoing it's Input Task at the Input Stream. When in an interim Layer, the Pallet may be either traversing a Link, moving through a Junction, or inside a Workstation. In the latter case, it is either waiting in the Buffer or undergoing its destination Task at the Machine. Finally, once in the Output Layer, the Pallet may either be traversing a Link, moving through a Junction, or inside an Output Station. When in an Output Station, the Pallet is either waiting in the Output Buffer or undergoing its Output Task in the Output Stream.

Action Scenarios

For the MPMS system model, we make the following observations before determining a suitable choice of action scenarios that characterize its event-driven evolution. First, the layered arrangement naturally separates the scenario evolution into corresponding layer-by-layer stages. Second, there is a pattern in the set of events possible in each stage due to the conceptually identical layout of each layer's components.

Guided by these observations, Figure 4-5 depicts a choice of action scenarios (identified by numbered states in the state diagram) appropriate for the MPMS system scenario. It spans how the scenario evolves when a single order is received and a single part type is processed. We argue that at our level of abstraction, there is not much loss of generality by explicitly excluding action scenarios where many pallets/orders are involved. We instead implicitly view additional pallets/orders as additional, *concurrent*, event-driven evolutions, each pallet/order experiencing the action scenario appropriate for its current position.

Table 4.1 lists the action scenarios identified for the general system scenario and their initiating and terminating events. We note that the events chosen to partition the action scenarios are inherent to the MPMS system scenario and will be present under any candidate architecture. Also included are action scenarios associated with two "exceptions" to the normal evolution of a system scenario. There is nothing inherently different about these action scenarios; only our interpretation as failures makes them an "exception." The types of failures we include are part failures and machine failures and a candidate architecture must support appropriate failure recovery mechanisms for both. A *part failure* refers to a pallet failing its inspection test upon leaving a workstation while a *machine failure* refers to a workstation becoming unusable. Exactly where the action scenarios for the exceptions occur depends on the system events internal to the regular action scenarios (i.e., depends on the choice of architecture).

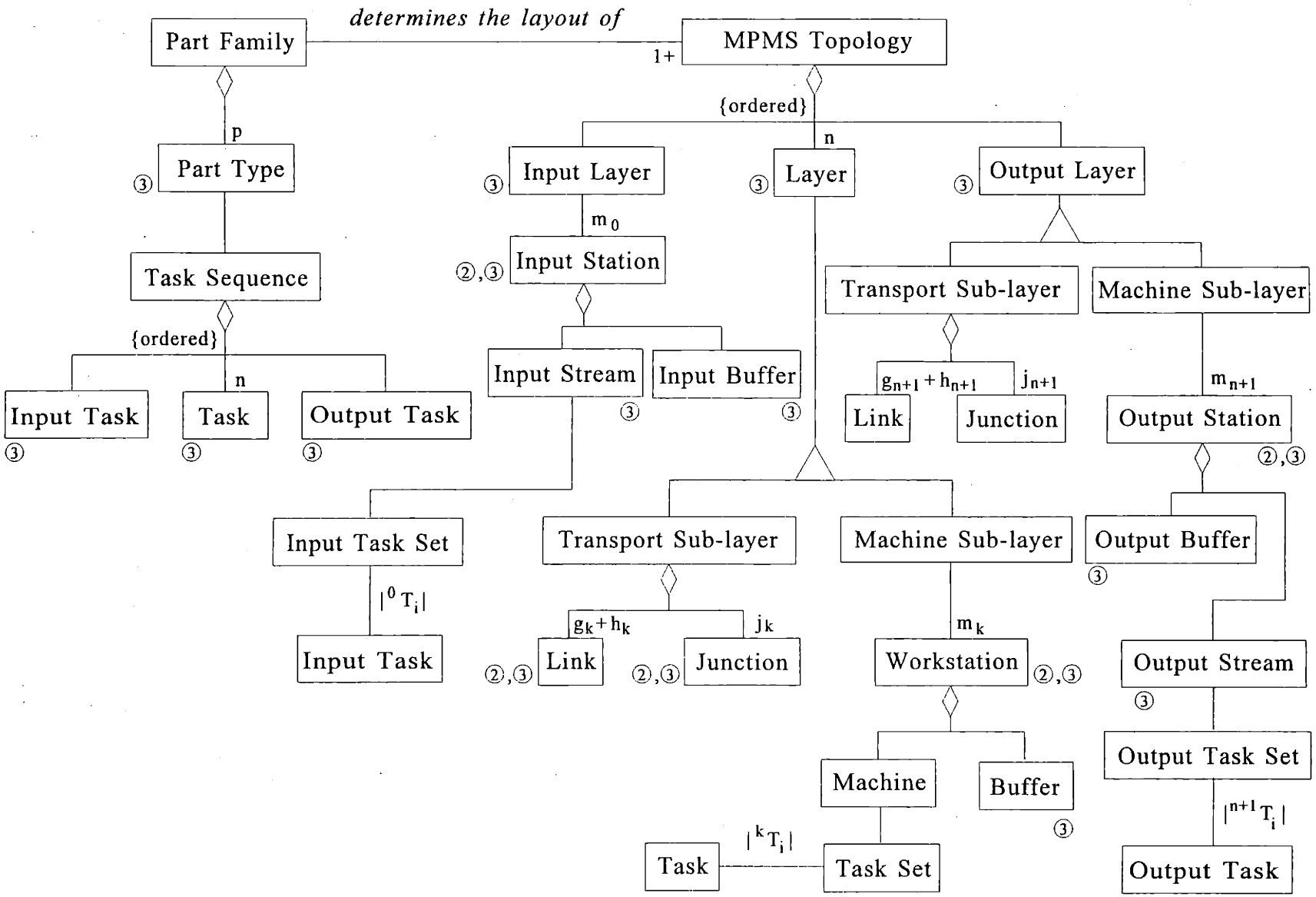


Figure 4-2: MPMS System Module, Sheet 1 - Product-Based Layout Relations

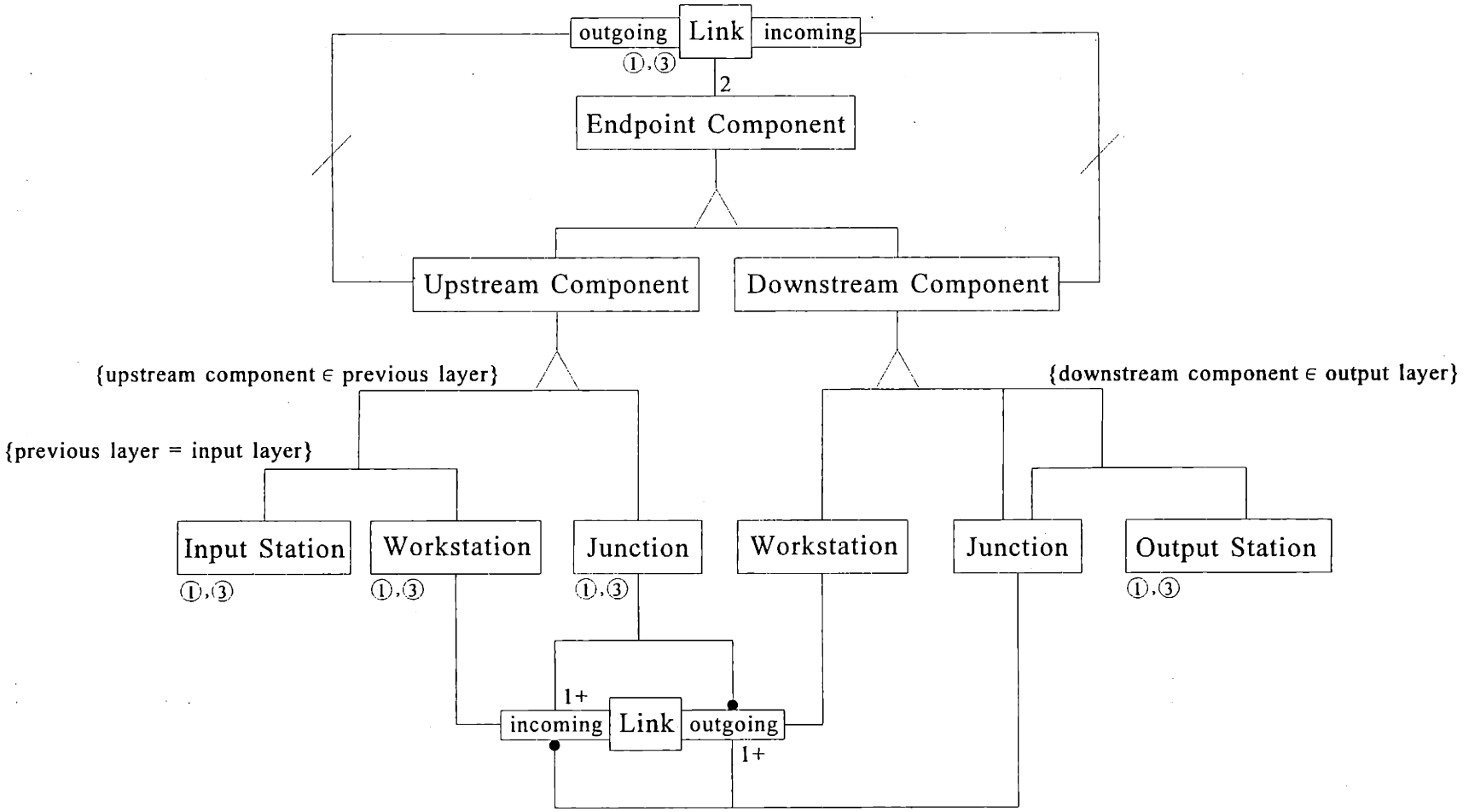


Figure 4-3: MPMS System Module, Sheet 2 - System Component Relations

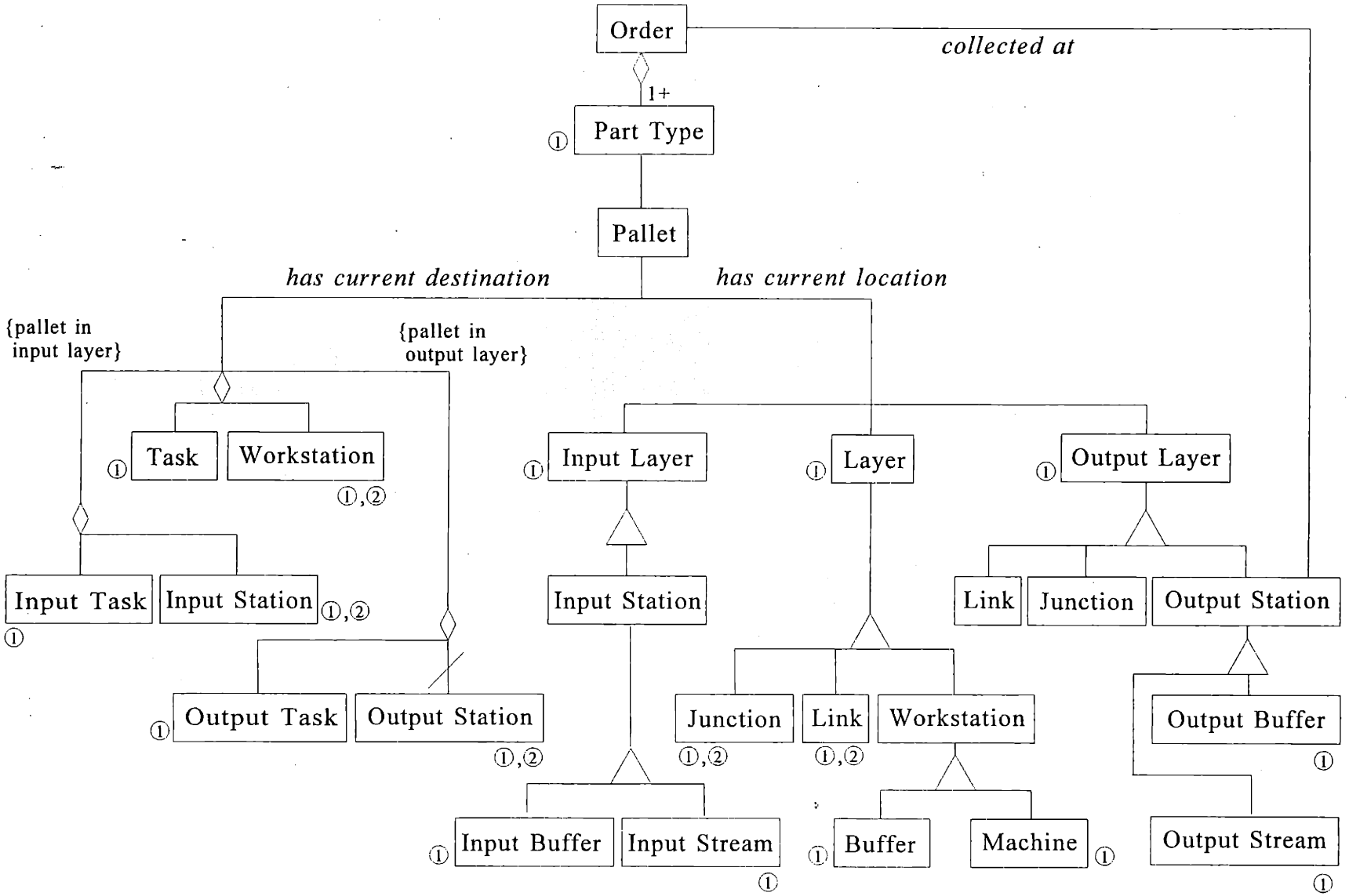


Figure 4-4: MPMS System Module, Sheet 3 - Order/Pallet Relations

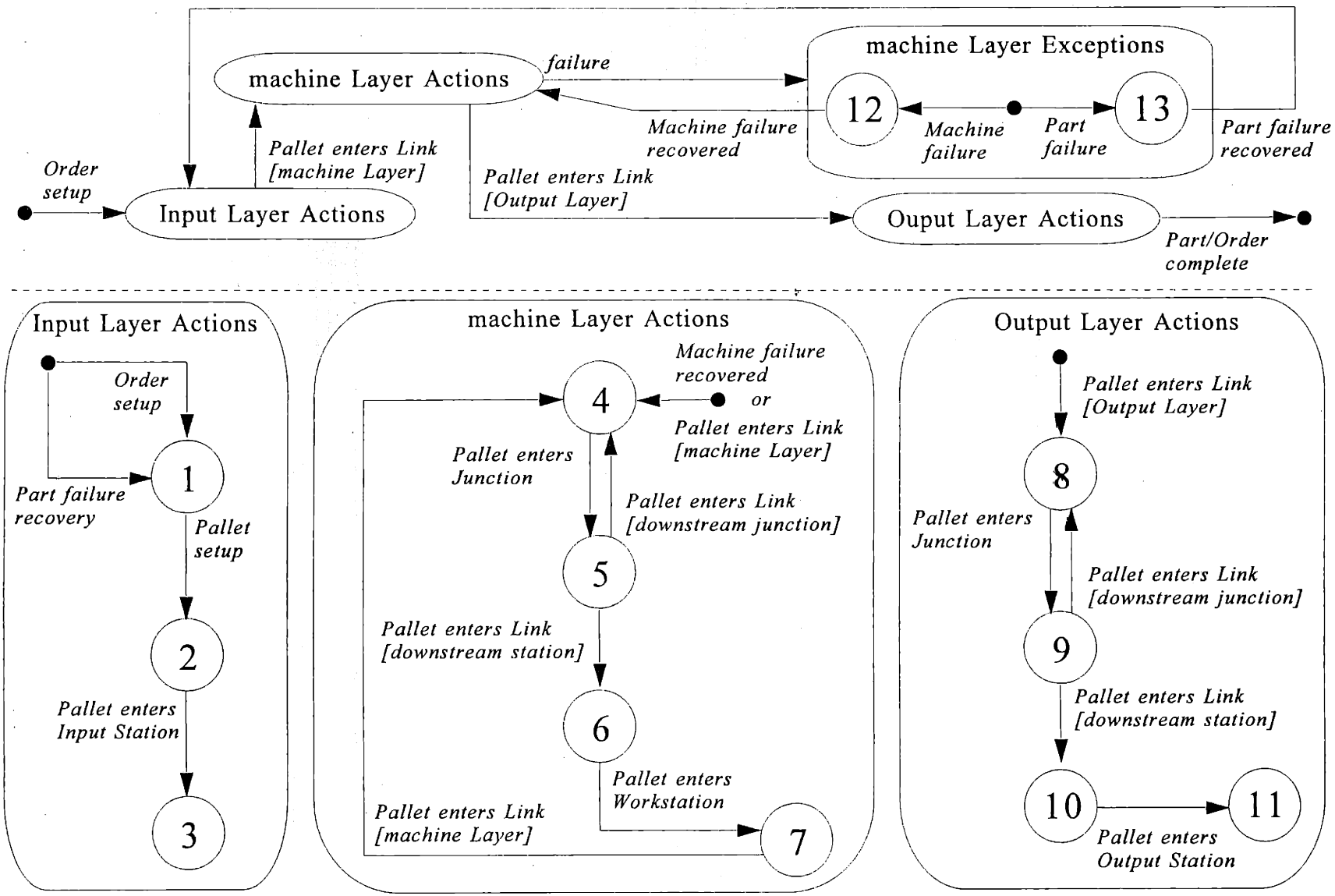


Figure 4-5: MPMS System Scenario - Action Scenarios

Action Scenario	Initiating Event	Terminating Event
Input Layer Actions		
1	Order setup	Pallet setup
2	Pallet setup	Pallet enters Input Station
3	Pallet enters Input Station	Pallet enters Link
machine Layer Actions		
4	Pallet enters Link	Pallet enters Junction
5	Pallet enters Junction	Pallet enters Link
6	Pallet enters Link	Pallet enters Workstation
7	Pallet enters Workstation	Pallet enters Link
Output Layer Actions		
8	Pallet enters Link	Pallet enters Junction
9	Pallet enters Junction	Pallet enters Link
10	Pallet enters Link	Pallet enters Output Station
11	Pallet enters Output Station	Part/Order complete
machine Layer Exceptions		
12	Part failure	Part failure recovered
13	Machine failure	Machine failure recovered

Table 4.1: Action Scenarios for MPMS System Scenario

4.4 Outline of Architectural Modeling Procedure

The scenario-based architectural modeling procedure we propose provides a structured approach to formulating and precisely describing candidate architectures. The initial step involves identifying a suitable conceptual framework, or system scenario, which captures typical features that a candidate architecture should account for. To this extent, we make use of the general MPMS system scenario formulated above. The next step for each candidate architecture involves composing an *architectural statement*, or a (usually incomplete) description of the candidate architecture within the conceptual framework of the specified system scenario. Using scenario-based object analysis techniques, for each architectural statement we obtain a *first-iteration architectural model*, or a precise abstraction of what the system components must do in order to achieve system correctness under the given architecture.

4.4.1 Sample MPMS System Scenario

To concisely describe a sample of the general MPMS system scenario described above, we use the terminology/notation introduced in Chapter 3 for parameterizing the MPMS system model. We simply assume an instance of a `MPMS topology` and an associated `part family`. As with any sample MPMS system scenario, the action scenarios of Table 4.1 are appropriate and characterize the event-driven evolution of the system scenario.

4.4.2 Architectural Statement

At our level of abstraction, we can characterize a distributed cooperative control architecture by three primary system-level features.

Decision Control: the set of decision rules between and within system components. Typically, it will center around a notion of a "cost function" that reflects the relative trade-offs between different decision policies.

Communication Capabilities: the "who talks to whom" relationship among components. Typically, there is an assumed reliable communication network that provides the channel for cooperation among components.

Information Management: the choice of relevant data, its interpretation within components, and methods for its exchange between the distributed components. Typically, this includes some preliminary assumptions regarding the presence of key data parameters, databases, and computation processes that the decision control and cooperation strategy rely upon.

4.4.3 Architectural Model

Once an architectural statement is formulated within an MPMS system scenario, the architectural model follows directly by applying scenario-based object analysis techniques. More specifically, each action scenario is *refined* (i.e, expressed by an ordered set of system events) to reflect the decision control, communication capabilities, and information exchange of the candidate architecture. Typically, this involves specifying a scenario script table for each action scenario.

The format of a scenario script table requires some explanation. Each line in a table corresponds to a single system-level event involving an exchange of messages between objects, or sub-systems. The sub-system that is sending a message in response to a system event is termed the *initiator*. The sub-system(s) that receives the message is termed the *participant* and may act upon a received message in one of several ways. The participant may save an internal piece of information, reply to update the initiator, initiate a message to update another participant, or any combination of the above. In any case, the function that is invoked by the message exchange is termed a *method*. Finally, listed under notes/assumptions are statements which aim to clarify the relations between the sequence of system-level events and inter-object messages.

From each scenario script table, we derive a set of relationships among system components consisting of three orthogonal but cross-linked descriptions:

Object Model: the separation into modular, inter-communicating sub-systems showing the data and procedures within each sub-system and identifying static relationships among them (a refined version of the modular description inherent to the MPMS system scenario identified in Figures 4-2, 4-3, and 4-4),

Dynamic Model: the dynamic behavior of sub-systems and how interactions among the sub-systems influence this behavior, and

Functional Model: the necessary information transfer between communicating sub-systems and computation capabilities required within each sub-system.

Taking the union of these derived sets of relationships yields a first-iteration model of the candidate architecture.

Chapter 5

Architectural Modeling Example

In this chapter, we propose two simple distributed cooperative control architectures by assuming a MPMS system environment and applying the architectural modeling procedure described in Chapter 4. Architecture A can be characterized as a "Pallet-driven/Station-decision" based architecture, meaning the distributed control strategy predominantly relies on the Pallets to initiate the cooperation required between components for a Station to make a system-level decision. Architecture B represents a "Pallet-driven/Guideway-decision" based architecture.

5.1 Sample MPMS System Scenario

As the conceptual basis in which to formulate architectural statements A and B, we use a sample of the general MPMS system scenario described in Chapter 4. We consider the given MPMS topology and the sample part family of Example 1 in Chapter 3. For convenience, the details of this example are summarized below and in Figure 5-1.

Part Family: $p = 3$; $x \in \{A, B, C\}$

$$\begin{aligned} Pt_A &\rightarrow Ts_A = (T_{A_i}, T_1, T_3, T_{A_o}) \\ Pt_B &\rightarrow Ts_B = (T_{B_i}, T_1, T_4, T_{B_o}) \\ Pt_C &\rightarrow Ts_C = (T_{C_i}, T_2, T_4, T_{C_o}) \end{aligned}$$

We use, as with any sample MPMS system scenario, the action scenarios identified in Table 4.1 to characterize the event-driven evolution.

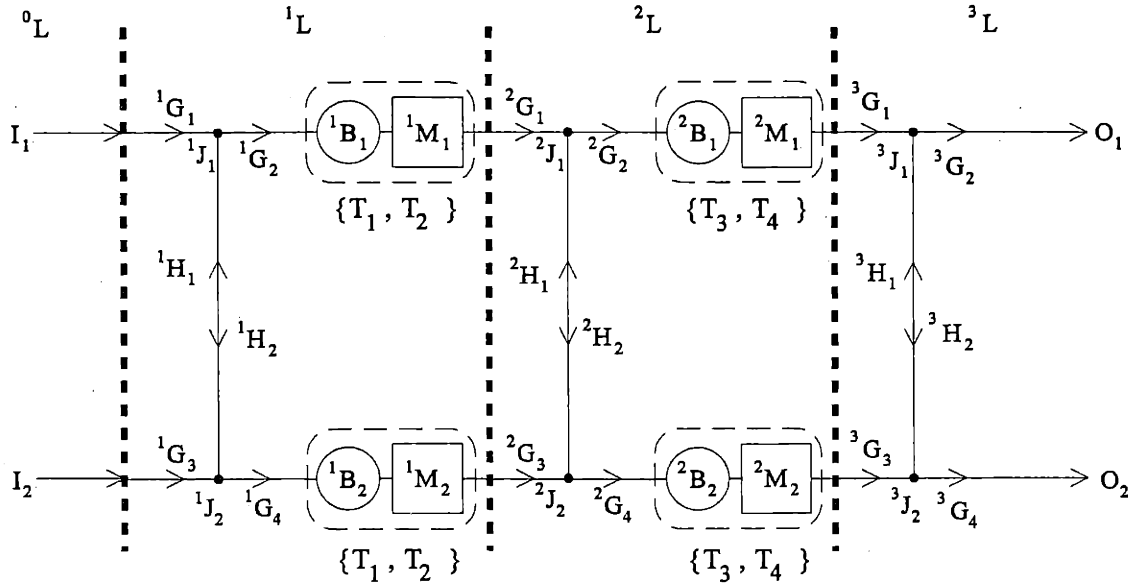


Figure 5-1: Sample MPMS Topology of Example 1, Chapter 3

5.2 Candidate Architectures

5.2.1 Architecture A

Architectural Statement

Decision Control (Pallet-driven/Station-decision based)

- Strategy: minimize a Station cost function $C_s(*)$

  $C_s(*)$ = cost of a Station to accept a Pallet relative to other Stations in the same Layer

- Pallets entering a Layer request Layer controller to interrogate Stations and, based on $C_s(*)$, choose an appropriate destination Station

Communication Capabilities

- Components exchange high-level messages over a reliable network
- "Who speaks-to-whom" structure of Figure 5-2 and, in addition,
  - No communication support between components in different Layers except between the Layer controllers themselves
  - No communication support between a Link and components other than its endpoints

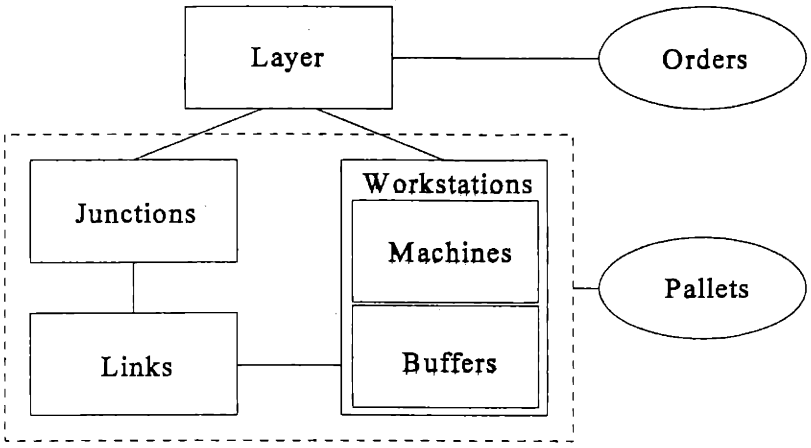


Figure 5-2: Architecture A, Communication Capabilities

- Pallets can communicate only with the sub-system in which they are currently located

Information Management

- A Layer records incoming orders in terms of the tasks, *required of that Layer*, corresponding to the Order's part type composition
- A Pallet is stamped with ID information and keeps track of current location, current task destination pertaining to its part type, and an assigned destination station
- Stations maintain cost parameters that affect computation of $C_s(*)$
- Junctions maintain a route map, or notion of which outgoing Links lead to which Stations within Layer
- Buffers maintain a list of Pallet arrivals and anticipated Pallet arrivals

Architectural Model

Tables A.1-13 of Appendix A contain the set of system events characterizing each action scenario, in the form of scenario script tables, under Architecture A. The first-iteration model of Architecture A is given by the union of the corresponding scenario-derived object analysis models and is also shown in Appendix A.

5.2.2 Architecture B

Architectural Statement

Decision Control (Pallet-driven/Guideway-decision based)

- Strategy: minimize a Guideway cost function $C_g(*)$

  $C_g(*)$ = cost of a Junction routing a Pallet onto an outgoing Link relative to other outgoing Links

- Stations periodically participate in a "bidding procedure" that reflects the Task preferences of the Stations; the results incorporated when computing $C_g(*)$
- Pallets entering a Link with a downstream Junction initiate Junction controller to interrogate outgoing Links and, based on $C_g(*)$, choose an appropriate outgoing Link

Communication Capabilities

- Same as for Architecture A

Information Management

- A Layer records incoming orders in terms of the tasks, *required of that* Layer, corresponding to the Order's part type composition
- A Pallet is stamped with ID information and keeps track of current location and task destination pertaining to its part type
- Links maintain cost parameters that affect computation of $C_s(*)$
- Junctions maintain a route map, or notion of which outgoing Links lead to which Stations within Layer, as well as the results of station bidding procedures
- Buffers maintain a list of Pallet arrivals

Architectural Model

Tables B.1-13 of Appendix B contain the set of system events characterizing each action scenario, in the form of scenario script tables, under Architecture B. Table B.14 denotes a sample bidding procedure which takes place in only a fraction of the action scenarios (namely #1, #2, and #13) under architecture B. The first-iteration model of Architecture B is given by the union of the corresponding scenario-derived object analysis models and is also shown in Appendix B.

5.3 Qualitative Analysis

All possibilities considered, the chosen pair of architectures are quite similar. By inspection of the architectural statements, a striking similarity is the choice of identical communication capabilities. Let us briefly discuss an example of how this fact is reflected in the architectural models. Consider the system-level events *inherent* to the system scenario's evolution (recall that these events occur under all architectures representable within the system scenario). Under each architecture, we recognize that the same components are involved in the same cooperation in response to an inherent event (*Pallet enters Link* in the models derived from Tables A.4 and B.4 for instance). In contrast, suppose Architecture A had a distinguishing communication capability where a pallet can communicate directly with any workstation in the layer. In this case, *Pallet enters Link* would involve direct cooperation between the pallet and workstations, resulting in differing cooperation between the architectures in response to an inherent system event.

Perhaps the most apparent difference between the pair of architectures is the choice of which sub-system calculates the cost function for each initiating pallet. In Architecture A, the calculation $C_s(*)$ is performed by the stations while in Architecture B, the calculation $C_g(*)$ is performed by the junctions. By contrasting the architectural models for action scenarios #1 and #4, we see that $C_s(*)$ incorporates *only* the state of the stations (machine and buffer) in the layer, whereas $C_g(*)$ incorporates the state of the local guideway *as well as* the (possibly outdated) state of the stations in the layer. The models for Table A.4 show that $C_s(*)$ uses the state of the stations at the instant that it is calculated. The models for Table B.4 show that $C_g(*)$ uses the state of the guideway and the "station preference list" at the instant that it is calculated. However, the models for Table B.1 indicate that the "station preference list" is generated only when a new order enters the system. Hence, the possibility exists that station information for $C_g(*)$ is outdated.

Based on the above discussion, we intuitively speculate on what conditions influence whether Architecture A or B is more suitable. Since under Architecture B the cost calculation incorporates state information of the local guideway in addition to state information of the stations, one might initially argue that Architecture B is always superior to A. However, recognizing the possibility of an outdated "station preference list," the additional information may actually have negative consequences. In cases of "short" inter-arrival times of incoming orders, relative to the dynamic evolution of the stations, the possibility for drastically outdated station information is reduced. In this case, it is reasonable to expect Architecture B to be superior to A.

Suppose it is determined that, under Architecture A, it is more likely for the decision control to "optimize" from the point-of-view of a workstation than under Architecture B (with a detailed analysis tool, this supposition could be experimentally tested). This implies that, in conditions where task change-over costs are likely to dominate over other costs, Architecture A is more suitable. For example, consider a set of orders that have a high variability in their part type composition, and thus high variability in the task composition required of each layer. In this case, workstations are more likely to require task change-overs between processing. In contrast, all other

things equal, suppose the orders are relatively homogeneous in terms of their part type composition. In this case costs such as transit time and link traffic may dominate over task change-over costs, implying Architecture B is more suitable.

5.4 Remarks

The simple example presented in this chapter supports the claim that the architectural modeling procedure proposed in Chapter 4 indeed captures, in addition to the flow and processing of parts, a precise abstraction of an architecture's system-level features: the decision control strategy, communication capabilities, and information management scheme. We also emphasize that the first-iteration models in the appendices are incomplete, since the example aims to clearly outline the steps involved rather than generate "ready-to-use" object analysis models. The proposed procedure is quite elaborate, reflecting the complexity inherent to modern information-intensive manufacturing. Unfortunately, understanding and/or synthesizing *complete* architectural models requires a considerable amount of effort in addition to a well-founded knowledge of object-oriented modeling techniques and formalisms.

Chapter 6

Conclusion

6.1 Summary

Technological advances in the areas of micro-computing, communications, and system science are leading the way for new capabilities in automated manufacturing systems. Emerging control area networks, consisting of intelligent components that can share information over a communication link without relying on a central controller, are motivating increasingly complex and large-scale factory-floor automation. Coupled with a rising interest for greater efficiency and flexibility, including improved robustness to failures and line augmentation, the future of factory-floor automation poses a major challenge in system development. In response to these trends, we addressed the potential that a distributed cooperative control environment provides for supporting new and advanced automation capabilities, while also incorporating the rising concern for improved system flexibility.

We began by introducing the general MPMS system topology that captures the typical features and complexity associated with a restricted class of discrete-part manufacturing processes. The overall MPMS system model, based upon the topology, logically supports a variety of system-level distributed cooperative control architectures. Building upon the system model, we proposed an architectural modeling procedure, guided by scenario-based object-oriented modeling techniques, for expressing a precise abstraction of the system-level features of a candidate architecture. Applying the procedure even in a simple example proved to be rather involved. However, the example supported the fact that a resulting architectural model indeed characterizes the corresponding architecture. Finally, we conclude that the true value of the approach is its naturally extension, by direct consequence of the object-oriented paradigm, to quantitative evaluation of alternative architectures through detailed simulation.

6.2 Suggestions for Further Research

The appropriate integration of many general areas of research is necessary to realize successful distributed cooperative control for modern automated manufacturing systems. In the overview given in Chapter 2, a number of these research areas were

identified. Specifically, many system-level issues related to decision control strategies, communication capabilities, and information management schemes in a distributed cooperative manufacturing environment require further investigation. At the device-level, even though standards for control area networks are emerging, research is still at its infant stages. It is difficult to precisely identify the progress in this area since much of the recent technology remains proprietary to industry developers. However, when addressing problems at the device-level, we suggest an emphasis on identifying techniques for translating system-level control decisions into efficient, time-critical, reactive automata which drive the distributed devices within a sub-system.

We now proceed with suggestions more directly related to this thesis. Regarding the MPMS system model, incorporating cycling and/or assembly operations would greatly extend its usefulness. Many modern manufacturing processes, such as those associated with the semiconductor industry, depend on these types of operations. An immediate value of the proposed architectural modeling procedure is its natural extension to object-oriented simulation. Simulation is currently the only viable approach for validating system correctness and obtaining a quantitative evaluation of system performance for complex manufacturing systems. A performance evaluation through simulation may address questions such as "Under a given architecture, what decision rules and scheduling policies result in the minimum order cycle times?" (see [33]) or, "Under what manufacturing situations does a candidate architecture outperform other candidate architectures?" Completing the object-oriented approach (i.e., developing graphical description languages for efficiently specifying systems and design/verification tools for analyzing pre-implemented systems) could, ultimately, be part of an overall Computer-Aided Control System Design (CACSD) [35] effort specific to the domain of manufacturing in a distributed cooperative environment.

Appendix A

Architecture A

In this appendix, we present a set of scenario script tables developed as an initial step towards obtaining the model for candidate Architecture A (pallet-driven/machine-decision based). Tables A.1-13 each correspond (as described in Chapter 3) to a particular *action scenario*, or thought experiment regarding the system scenario evolution under Architecture A. Following the tables are shown the corresponding scenario-derived object analysis models that comprise the first-iteration model of Architecture A.

Initiator	Event	Participant	Method	Notes/Assumptions
Sample MPMS System	order setup	Order Or_1	create	system receives customer order; begin action scenario #1
Order Or_1	communicate order entry	Layers ^0L, ^1L, ^2L, ^3L	introduce	layers save required tasks pertaining to order's part type composition
Order Or_1	request output station assignment	Layer ^3L (output layer)	request	order needs output station assignment
Layer ^3L (output layer)	query output stations for order assignment	Output Stations O_1, O_2	query	---
Output Stations O_1, O_2	compute cost and return result	Layer ^3L (output layer)	cost	possible factors: output stream backlog on previous orders, order priority
Layer ^3L (output layer)	communicate choice for output station assignment	Output Station O_1, Order Or_1	choose	O_1 returns lowest result; output station saves order assignment
Layer ^0L (input layer)	pallet setup	Input Stations I_1, I_2	query	end action scenario #1; begin action scenario #2

Table A.1: Architecture A, Action Scenario #1

Initiator	Event	Participant	Method	Notes/Assumptions
Layer ^{0}L (input layer)	pallet setup	Input Stations I_1, I_2	query	end action scenario #1; begin action scenario #2
Input Stations I_1, I_2	compute cost and return result	Layer ^{0}L (input layer)	cost	possible factors: input stream setup/backlog, pallet/order priority, part
Layer ^{0}L (input layer)	communicate choice for input station assignment	Input Station I_1	choose	I_1 returns lowest result; input station saves pallet setup
Input Station I_1	prepare raw pallet	Pallet P_{A1}	prepare	pallet tagged with part type A, task sequence TS_A, member order Or_1
Pallet P_{A1}	raw pallet enters input station	Input Buffer Ib_1	introduce	end action scenario #2; begin action scenario #3

Table A.2: Architecture A, Action Scenario #2

Initiator	Event	Participant	Method	Notes/Assumptions
Pallet P_{Al}	raw pallet enters input station	Input Buffer Ib_1	introduce	end action scenario #2; begin action scenario #3
Input Buffer Ib_1	communicate pallet waiting for input stream processing	Input Stream Is_1	update	input buffer empty prior to pallet arrival
Input Stream Is_1	communicate readiness for next pallet in input buffer queue	Input Buffer Ib_1	ready	input stream idle prior to pallet arrival
Pallet P_{Al}	raw pallet enters input stream	Input Stream Is_1	introduce	input buffer takes pallet off its record; input stream sets up for required input task
Input Stream Is_1	perform input task on raw pallet	Pallet P_{Al}	perform	raw pallet loaded with raw material pertaining to particular part type
Input Stream Is_1	communicate successful task completion on pallet	Pallet P_{Al} Layer ^0L	complete	layer updates its record; pallet updates its destination task
Input Stream Is_1	communicate readiness for next raw pallet in input buffer queue	Input Buffer Ib_1	ready	---
Pallet P_{Al}	pallet enters link (with downstream junction)	Link $^1G_1$	introduce	end action scenario #3; begin action scenario #4 (Layer 1)

Table A.3: Architecture A, Action Scenario #3

Initiator	Event	Participant	Method	Notes/Assumptions
Pallet P_{A1}	pallet enters link (with downstream junction)	Link $^1G_1$	introduce	end action scenario #3; begin action scenario #4 (Layer 1)
Link $^1G_1$	communicate pallet entry	Junction $^1J_1$	update	link has downstream junction
Junction $^1J_1$	determine if pallet needs workstation assignment	---	detect	---
Junction $^1J_1$	request workstation assignment	Layer ^1L	request	pallet needs workstation assignment
Layer ^1L	query accessible workstations for pallet assignment	Workstations $^1W_1$,$^1W_2$	query	layer knows which workstations are accessible from junction
Workstations $^1W_1$,$^1W_2$	compute cost and return result	Layer ^1L	cost	possible factors: queue state, task completion times, setup transitions, pallet priority
Layer ^1L	communicate choice for workstation assignment	Junction $^1J_1$, Workstation $^1W_1$	choose*[1]	$^1W_1$ returns lowest result; junction saves assigned station
Workstation $^1W_1$	communicate anticipated pallet arrival	Buffer $^1B_1$	anticipate	buffer keeps queue saves anticipated pallet arrivals
Pallet P_{A1}	pallet enters junction	Junction $^1J_1$	introduce	end action scenario #4; begin action scenario #5

*[1] If a machine failure has occurred, cost returns an "infinite" value indicating to layer that the machine is down. end action scenario #4; begin action scenario #13

Table A.4: Architecture A, Action Scenario #4

Initiator	Event	Participant	Method	Notes/Assumptions
Pallet P_{A1}	pallet enters junction	Junction $^1J_1$	introduce	end action scenario #4; begin action scenario #5
Junction $^1J_1$	tell pallet its destination workstation and route it	Pallet P_{A1}	route	workstation assignment completed before pallet enters junction
Pallet P_{A1}	pallet enters link (with downstream workstation)	Link $^1G_2$	introduce	end action scenario #5; begin action scenario #6*[2]

*[2] If Workstation $^1W_2$ rather than $^1W_1$ had been chosen, then pallet would have been routed to Link $^1H_2$ which has a downstream junction, and action scenario #4 rather than #6 would begin.

Table A.5: Architecture A, Action Scenario #5

Initiator	Event	Participant	Method	Notes/Assumptions
Pallet P_{A1}	pallet enters link (with downstream workstation)	Link $^1G_2$	introduce	end action scenario #5; begin action scenario #6
Link $^1G_2$	communicate pallet entry	Workstation $^1W_1$	update	link has downstream workstation
Workstation $^1W_1$	communicate confirmed pallet arrival	Buffer $^1B_1$	confirm	anticipated pallet arrival now saved as confirmed pallet arrival
Pallet P_{A1}	pallet enters workstation	Buffer $^1B_1$	introduce	buffer savess arrival; end action scenario #6; begin action scenario #7

Table A.6: Architecture A, Action Scenario #6

Initiator	Event	Participant	Method	Notes/Assumptions
Pallet P_{A1}	pallet enters workstation	Buffer $^1B_1$	introduce	end action scenario #6; begin action scenario #7
Buffer $^1B_1$	communicate pallet waiting for machine processing	Machine $^1M_1$	update	buffer empty prior to pallet arrival
Machine $^1M_1$	communicate readiness for next pallet in buffer queue	Buffer $^1B_1$	ready	machine idle prior to pallet arrival
Pallet P_{A1}	pallet enters machine	Machine $^1M_1$	introduce	buffer takes pallet off its record; machine sets up for required task
Machine $^1M_1$	perform required task on pallet	Pallet P_{A1}	perform	machine moves pallet into inspection station area upon completed task
Machine $^1M_1$	inspect part on pallet	Pallet P_{A1}	inspect*[3]	part on pallet passes inspection test
Machine $^1M_1$	communicate successful task completion on pallet	Pallet P_{A1} Layer ^1L	complete	layer saves its record; pallet saves its destination task
Machine $^1M_1$	communicate readiness for next pallet in buffer queue	Buffer $^1B_1$	ready	---
Pallet P_{A1}	pallet enters link (with downstream junction)	Link $^2G_1$	introduce	end action scenario #7 (Layer 1); begin action scenario #4 (Layer 2)

*[3] If a part failure has occurred, inspection fails, end action scenario #7; begin action scenario #12.

Table A.7: Architecture A, Action Scenario #7

Initiator	Event	Participant	Method	Notes/Assumptions
Pallet P_{A1}	pallet enters link (with downstream junction)	Link $^3G_1$	introduce	end action scenario #7 (layer 2); begin action scenario #8
Link $^3G_1$	communicate pallet entry	Junction $^3J_1$	update	link has downstream junction
Junction $^3J_1$	determine if pallet needs output station assignment	---	detect	---
Junction $^3J_1$	request output station assignment	Layer ^3L (output layer)	request	pallet needs output station assignment
Layer ^3L (output layer)	query output stations for pallet assignment	Output Stations O_1, O_2	query	---
Output Station O_1	compute cost and return result	Layer ^3L (output layer)	cost	O_1 assigned to order containing pallet
Layer ^3L (output layer)	communicate destination output station	Output Station O_1, Junction $^3J_1$	choose	O_1 returns zero cost; junction saves assigned output station
Output Station O_1	communicate anticipated pallet arrival	Output Buffer Ob_1	anticipate	output buffer queue saves anticipated pallet arrivals
Pallet P_{A1}	pallet enters junction	Junction $^3J_1$	introduce	end action scenario #8; begin action scenario #9

Table A.8: Architecture A, Action Scenario #8

Initiator	Event	Participant	Method	Notes/Assumptions
Pallet P_{A1}	pallet enters junction	Junction $^3J_1$	introduce	end action scenario #8; begin action scenario #9
Junction $^3J_1$	tell pallet its destination output station and route it	Pallet P_{A1}	route	output station assignment completed before pallet enters junction
Pallet P_{A1}	pallet enters link (with downstream output station)	Link $^3G_2$	introduce	end action scenario #9; begin action scenario #10*[4]

*[4] If Output Station O_2 rather than O_1 had been chosen, then pallet would have been routed to Link $^3H_2$ which has a downstream junction, and action scenario #8 rather than #10 would begin.

Table A.9: Architecture A, Action Scenario #9

Initiator	Event	Participant	Method	Notes/Assumptions
Pallet P_{A1}	pallet enters link (with downstream output station)	Link $^3G_2$	introduce	end action scenario #9; begin action scenario #10
Link $^3G_2$	communicate pallet entry	Output Station O_1	update	link has downstream output station
Output Station O_1	communicate confirmed pallet arrival	Output Buffer Ob_1	confirm	anticipated pallet arrival now saved as confirmed pallet arrival
Pallet P_{A1}	pallet enters output station	Output Buffer Ob_1	introduce	end action scenario #10; begin action scenario #11

Table A.10: Architecture A, Action Scenario #10

Initiator	Event	Participant	Method	Notes/Assumptions
Pallet P_{A1}	pallet enters output station	Output Buffer Ob_1	introduce	end action scenario #10; begin action scenario #11
Output Buffer Ob_1	communicate pallet waiting for output stream processing	Output Stream Os_1	update	output buffer empty prior to pallet arrival
Output Stream Os_1	communicate readiness for next pallet in output buffer queue	Output Buffer Ob_1	ready	output stream idle prior to pallet arrival
Pallet P_{A1}	pallet enters output stream	Output Stream Os_1	introduce	output buffer takes pallet off its record; output stream sets up for required task
Output Stream Os_1	perform output task on pallet	Pallet P_{A1}	perform	unload finished part from pallet
Output Stream Os_1	communicate successful part type completion	Layer ^3L (output layer)	complete	layer saves its record
Output Stream Os_1	communicate readiness for next pallet in output buffer queue	Output Buffer Ob_1	ready	--
Layer ^3L (output layer)	communicate part completion	Order Or_1	part_complete	order updates its record end action scenario #11*[5]
*[5] if completed part is final part in order, scenario continues...				
Layer ^3L (output layer)	communicate order completion	Output Station O_1, Order Or_1	order_complete	end action scenario #11;

Table A.11: Architecture A, Action Scenario #11

Initiator	Event	Participant	Method	Notes/Assumptions
Machine $^1M_1$	inspect part on pallet	Pallet P_{A1}	inspect*[6]	part on pallet fails inspection test
Machine $^1M_1$	communicate unsuccessful process completion on pallet	Layer ^1L, Pallet P_{A1}	incomplete	pallet saves its destination task - "fix fault"
layers and order coordinate for part failure recovery				
Layer ^1L	communicate need for faulty pallet replacement	Order Or_1	replace	---
Order Or_1	communicate replacement order entry	Layers ^0L, ^1L, ^2L, ^3L	update	replacement order placed with higher priority;
Layer ^0L (input layer)	pallet setup	Input Stations I_1, I_2	query	pallet requested with higher priority; begin action scenario #2 (with new pallet)
faulty pallet is redirected to a "fault bin"				
Machine $^1M_1$	communicate readiness for next pallet in buffer queue	Buffer $^1B_1$	ready	---
Pallet P_{A1}	pallet enters link (with downstream junction)	Link $^2G_1$	introduce (faulty)	pallet has destination station of "fault bin"; begin action scenario #4 (Layer 2)
end action scenario #12				

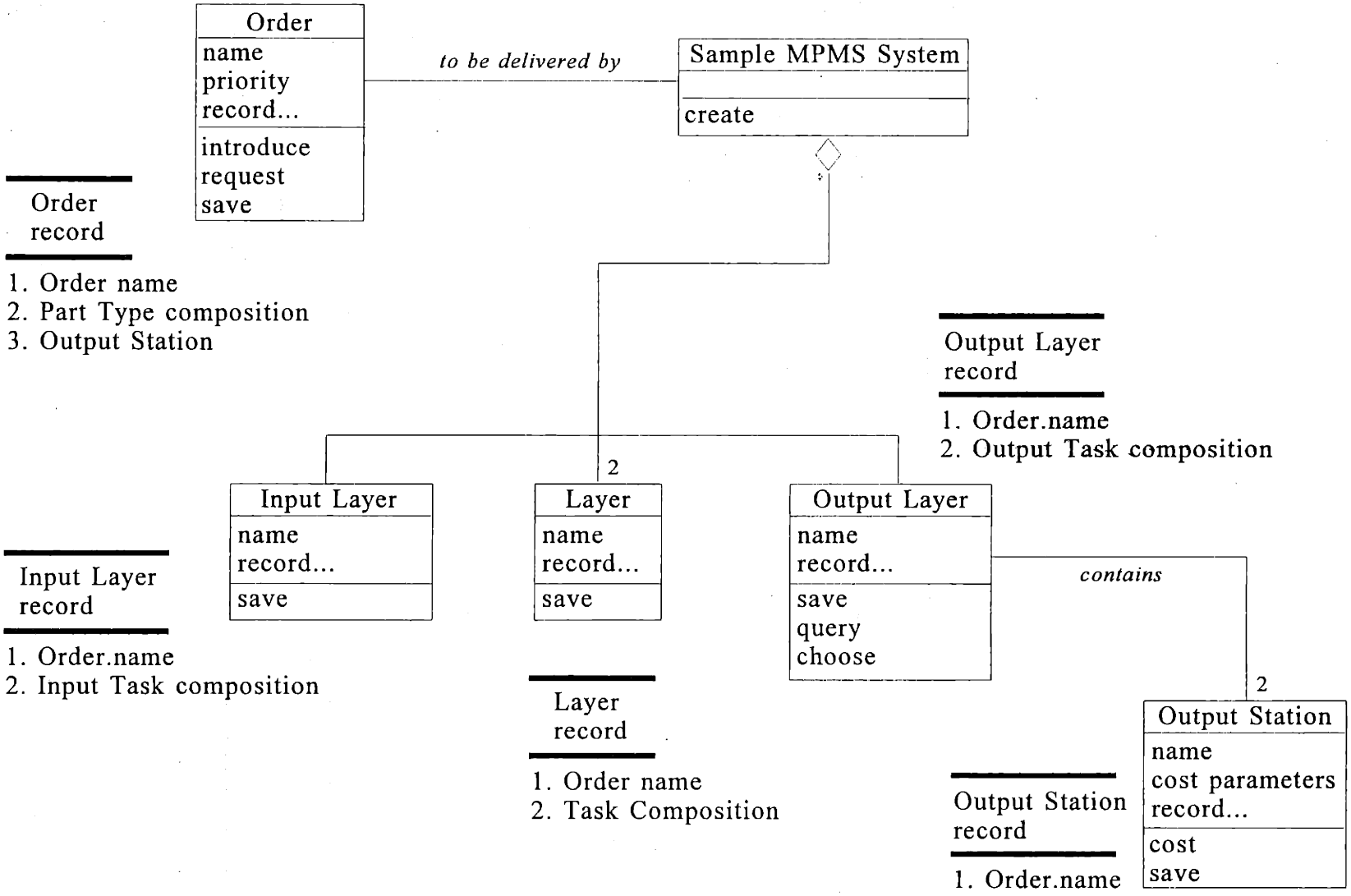
*[6] Action scenario #7 ends early due to part failure (inspection test fails), end action scenario #7; begin action scenario #12.

Table A.12: Architecture A, Action Scenario #12

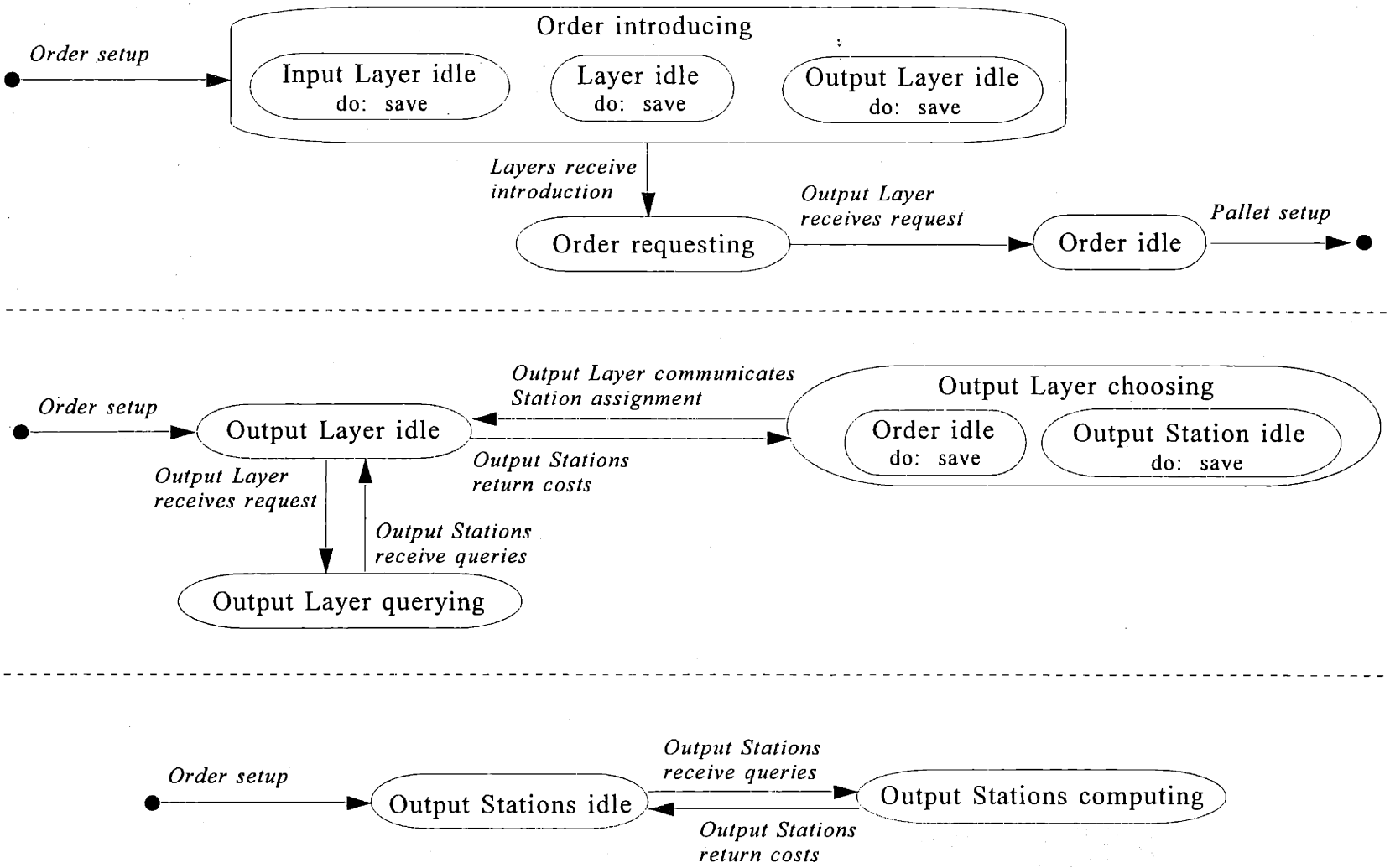
Initiator	Event	Participant	Method	Notes/Assumptions
Layer ^1L	communicate choice for workstation assignment	Workstation $^1W_2$, Junction $^1J_1$	choose*[7]	$^1W_1$ returns "infinite" result $^1W_2$ returns lowest result
Layer ^1L	layer communicates machine failure to system	Simple MPMS Topology	update	system reconfigures topology to account for effect of machine failure
Workstation $^1W_2$	communicate anticipated pallet arrival	Buffer $^1B_2$	anticipate	buffer saves anticipated pallet arrivals
Pallet P_{A1}	pallet enters junction	Junction $^1J_1$	introduce	end action scenario #13; begin action scenario #5

*[7] Action scenario #4 ends early due to machine failure ($^1W_1$ returns an "infinite" cost); begin action scenario #13.

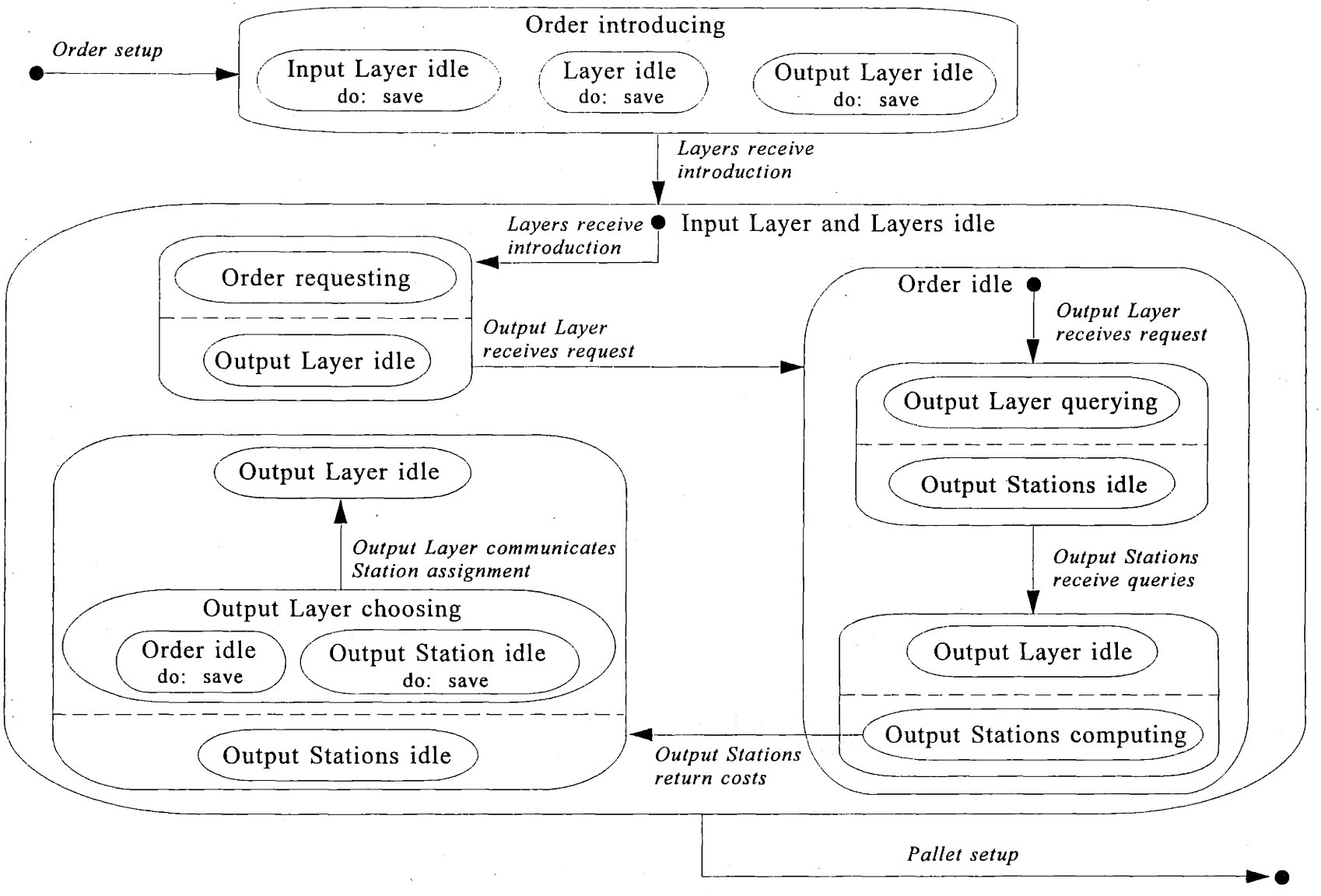
Table A.13: Architecture A, Action Scenario #13



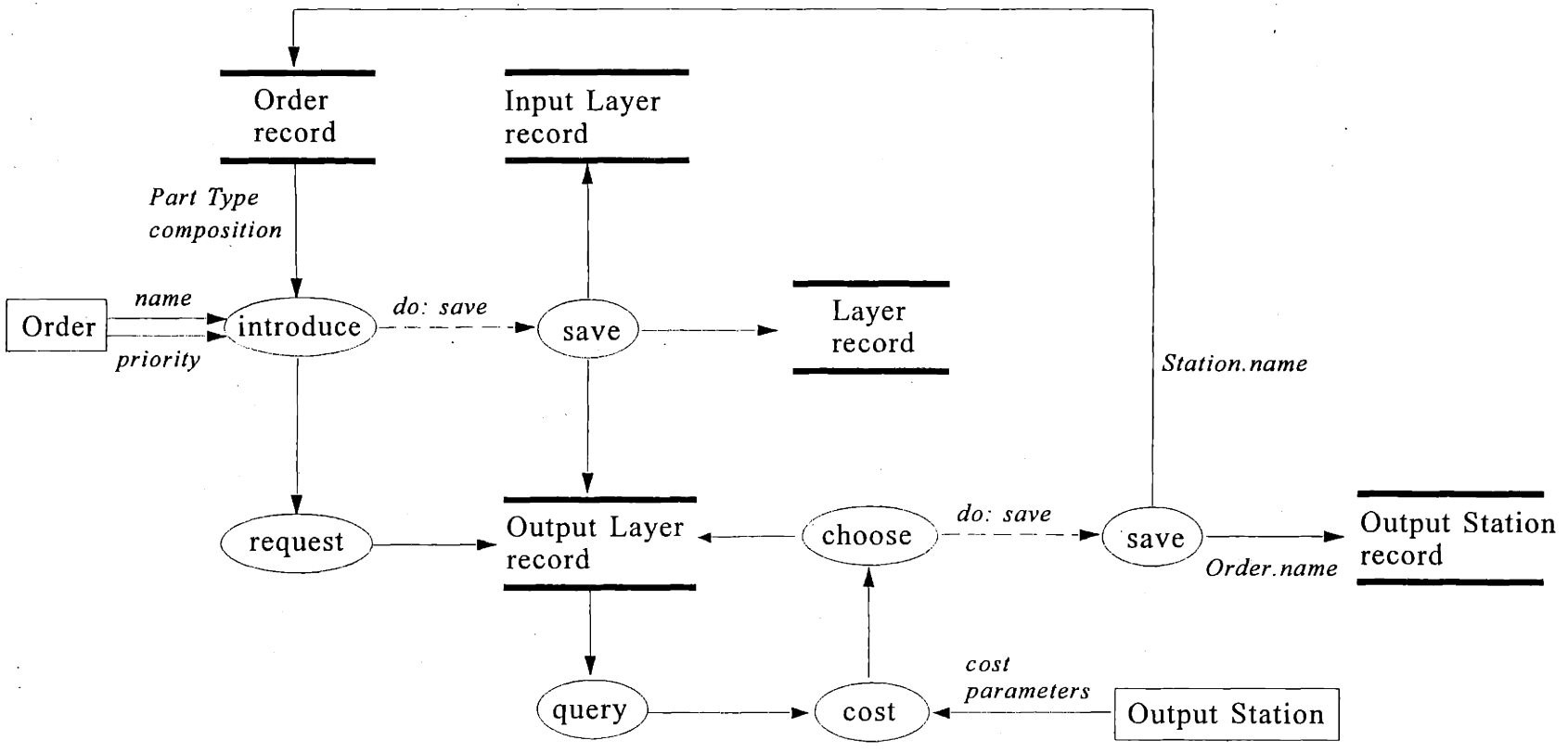
Architecture A, Action Scenario #1: Object Model



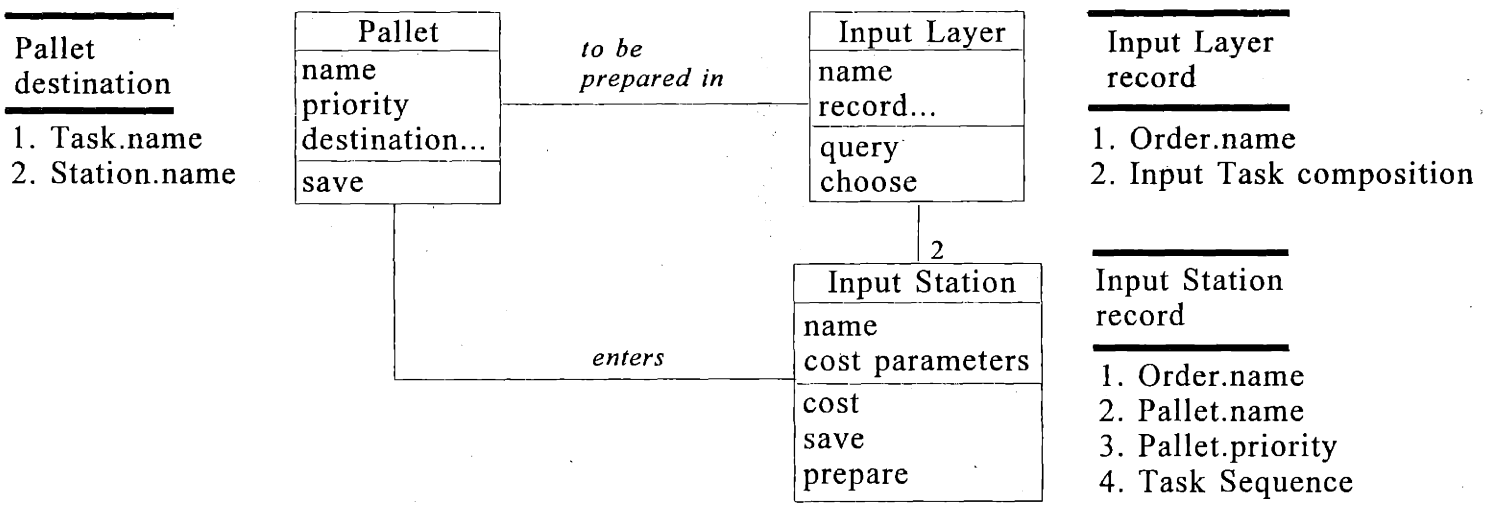
Architecture A, Action Scenario #1: Dynamic Model, Sheet 1



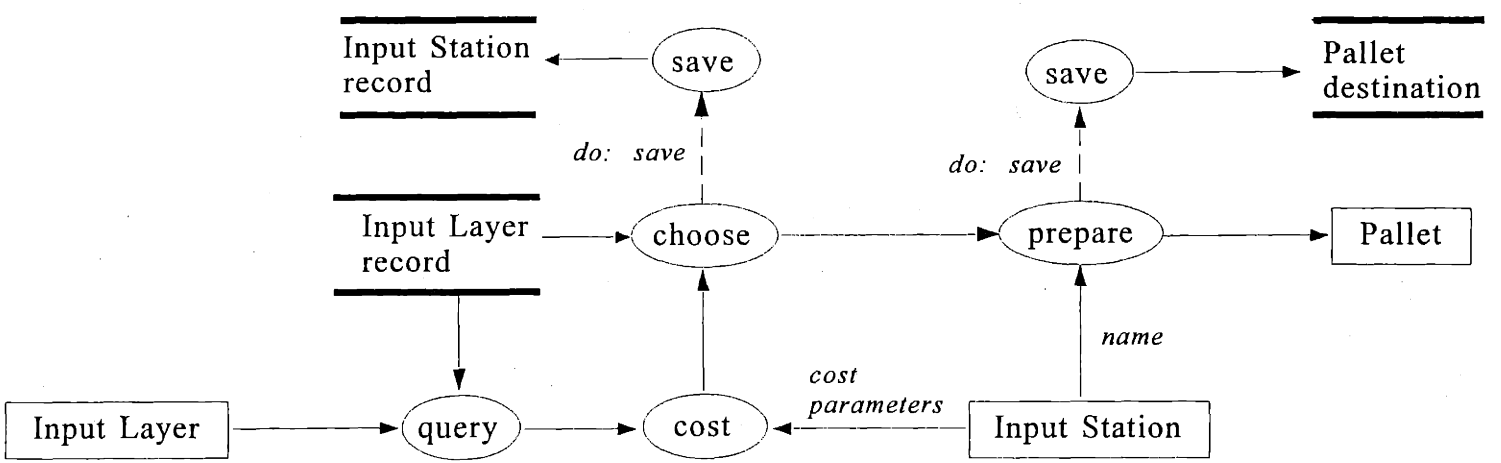
Architecture A, Action Scenario #1: Dynamic Model, Sheet 2



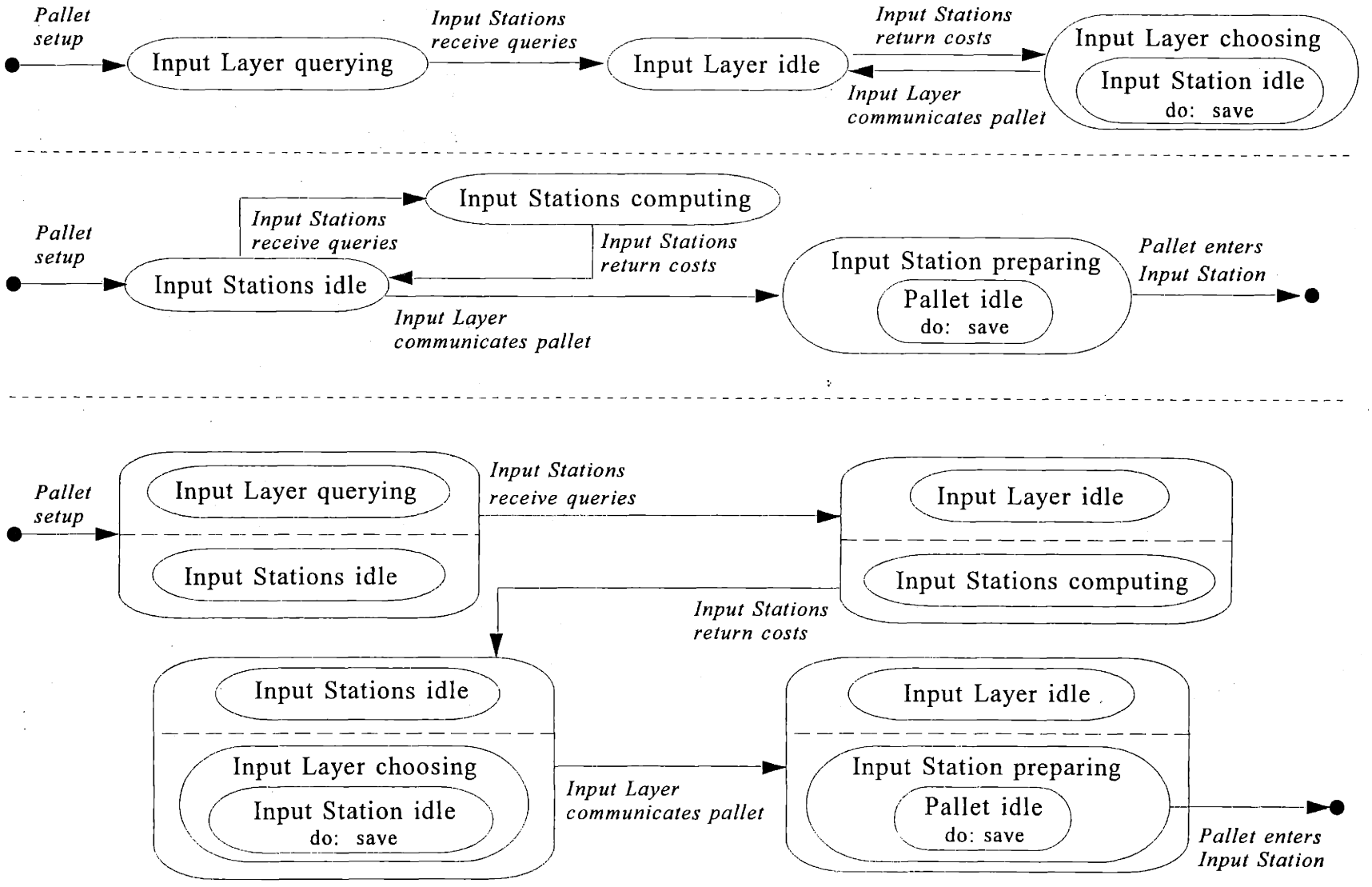
Architecture A, Action Scenario #1: Functional Model



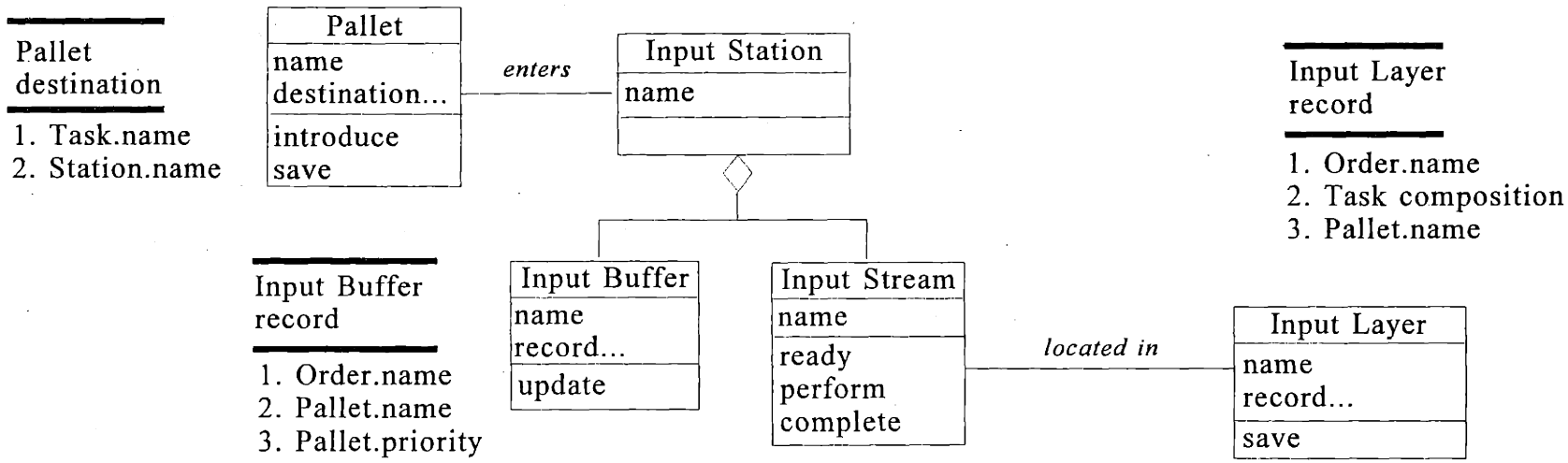
Architecture A, Action Scenario #2: Object Model



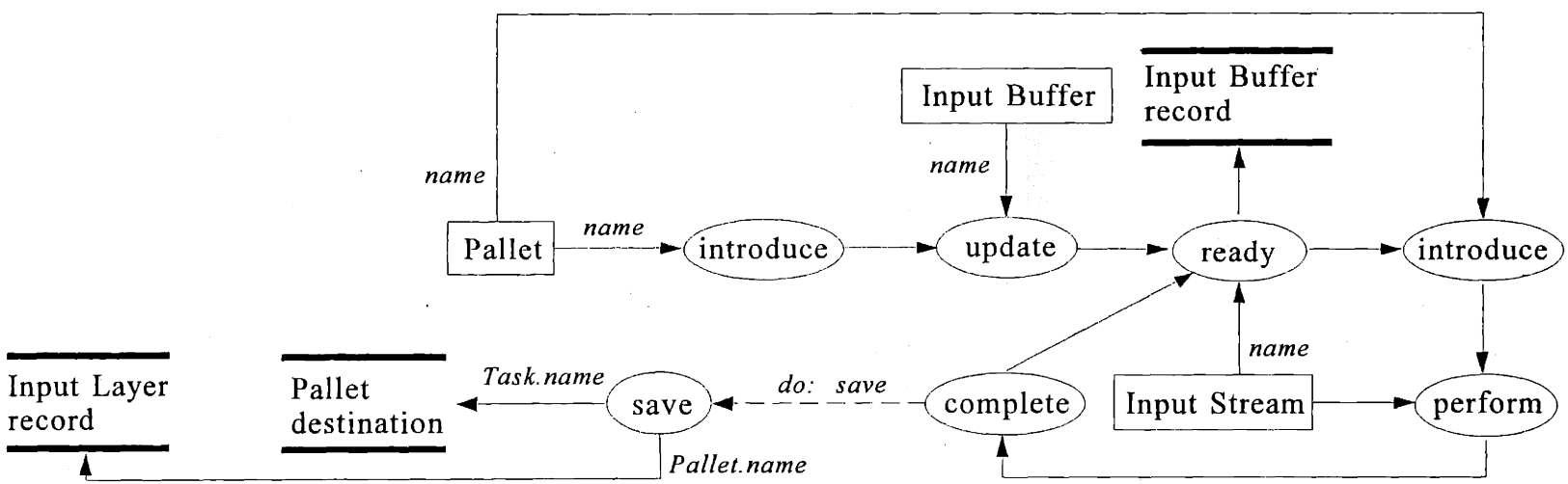
Architecture A, Action Scenario #2: Functional Model



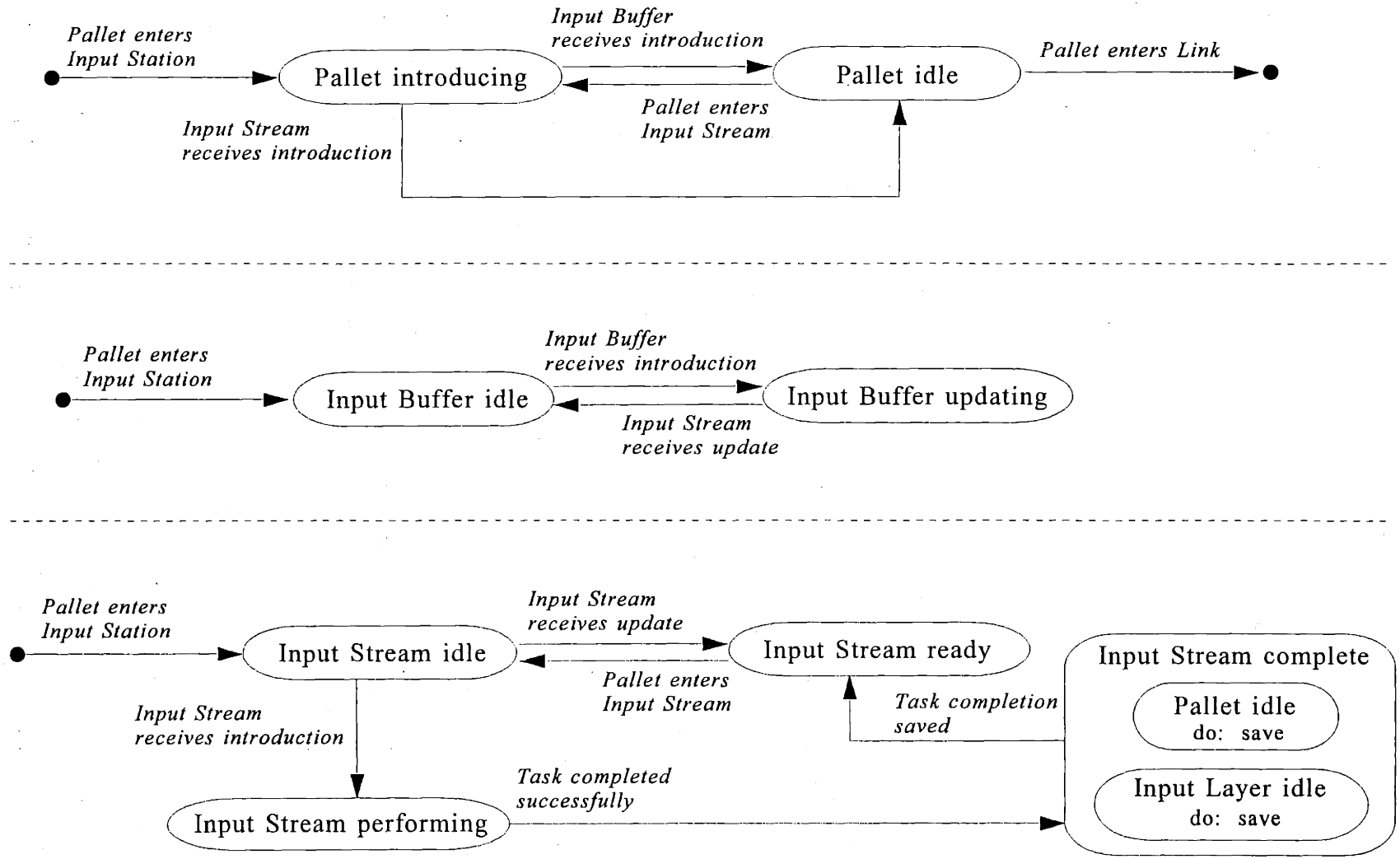
Architecture A, Action Scenario #2: Dynamic Model



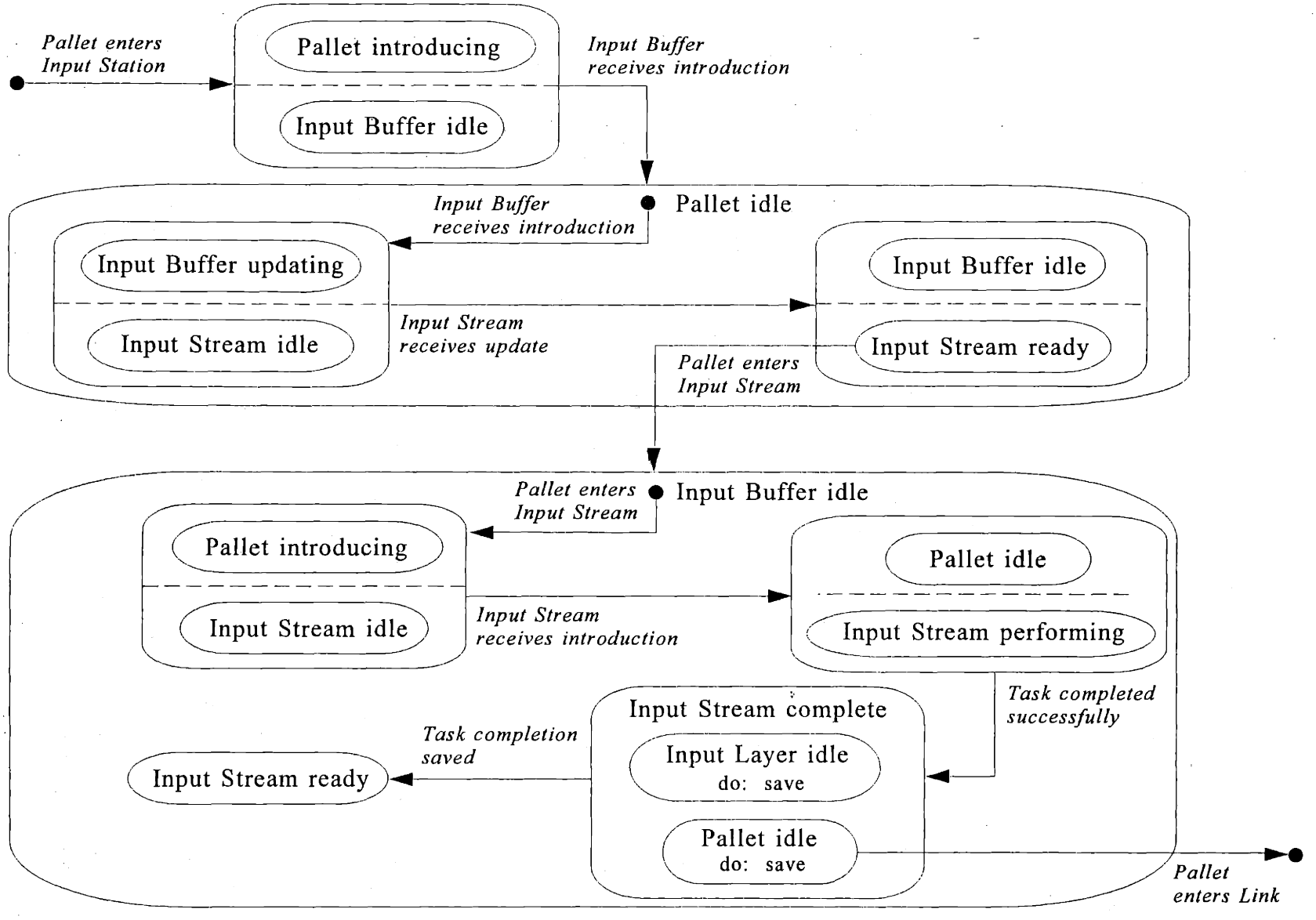
Architecture A, Action Scenario #3: Object Model



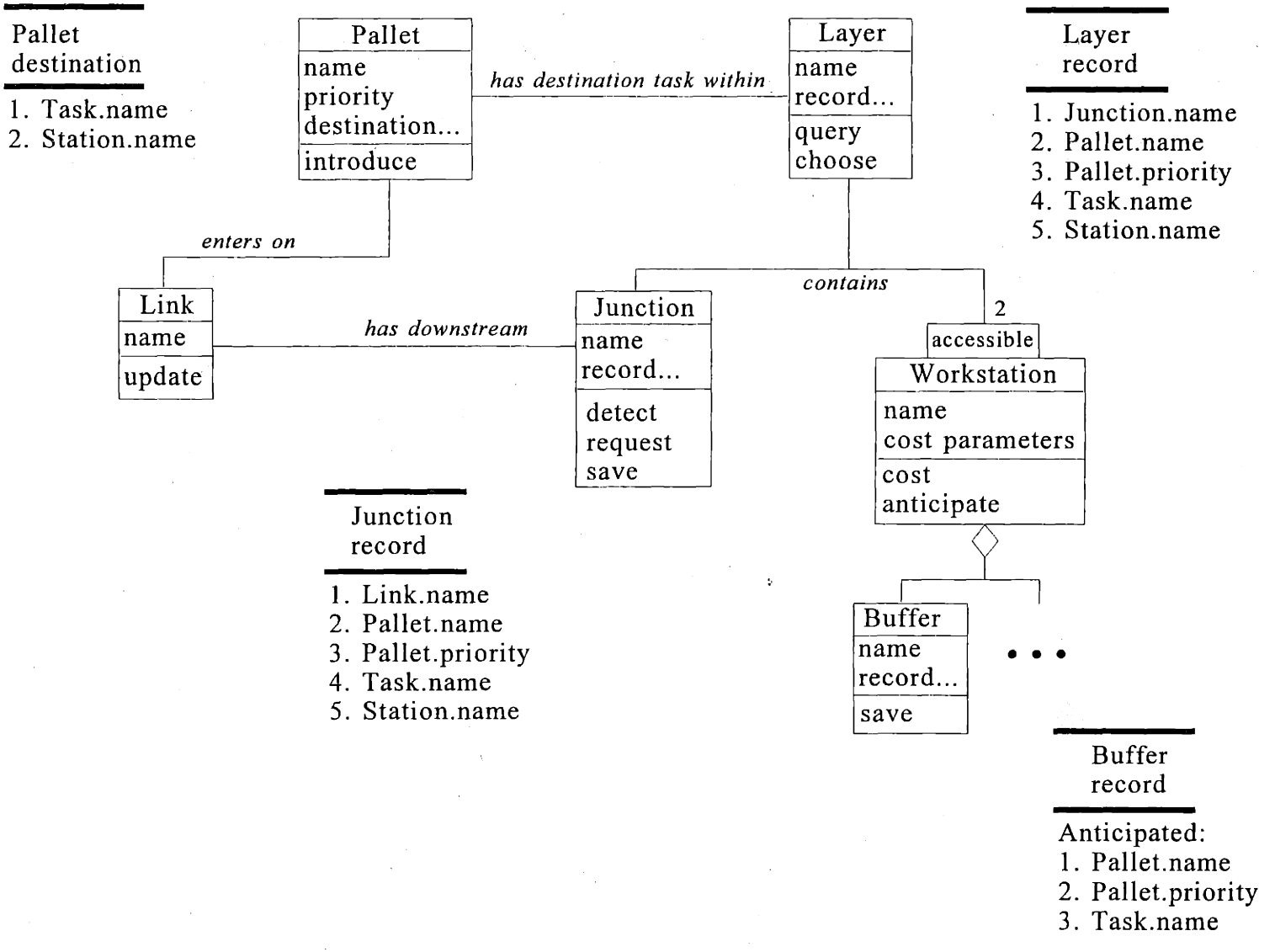
Architecture A, Action Scenario #3: Functional Model



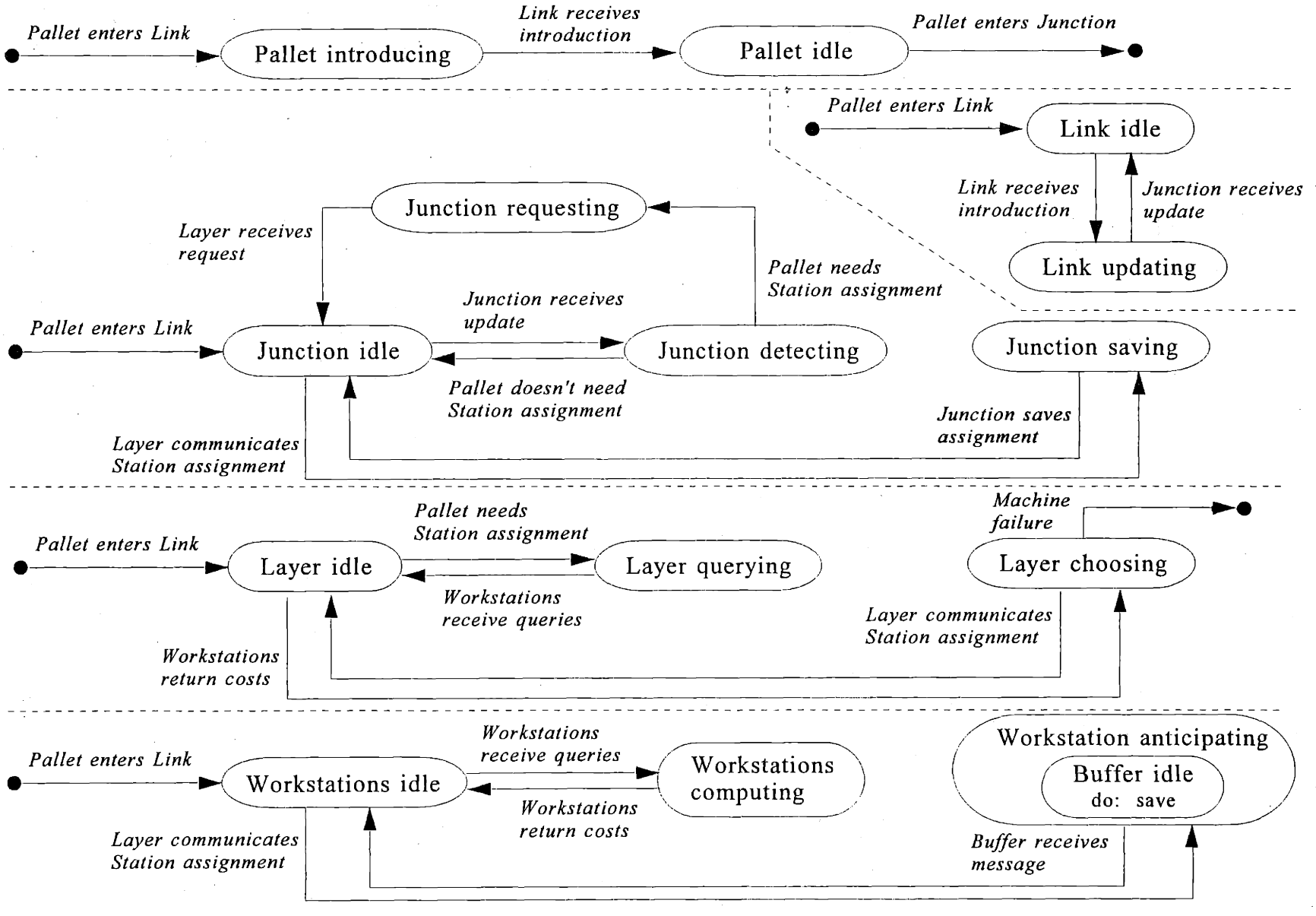
Architecture A, Action Scenario #3: Dynamic Model, Sheet 1



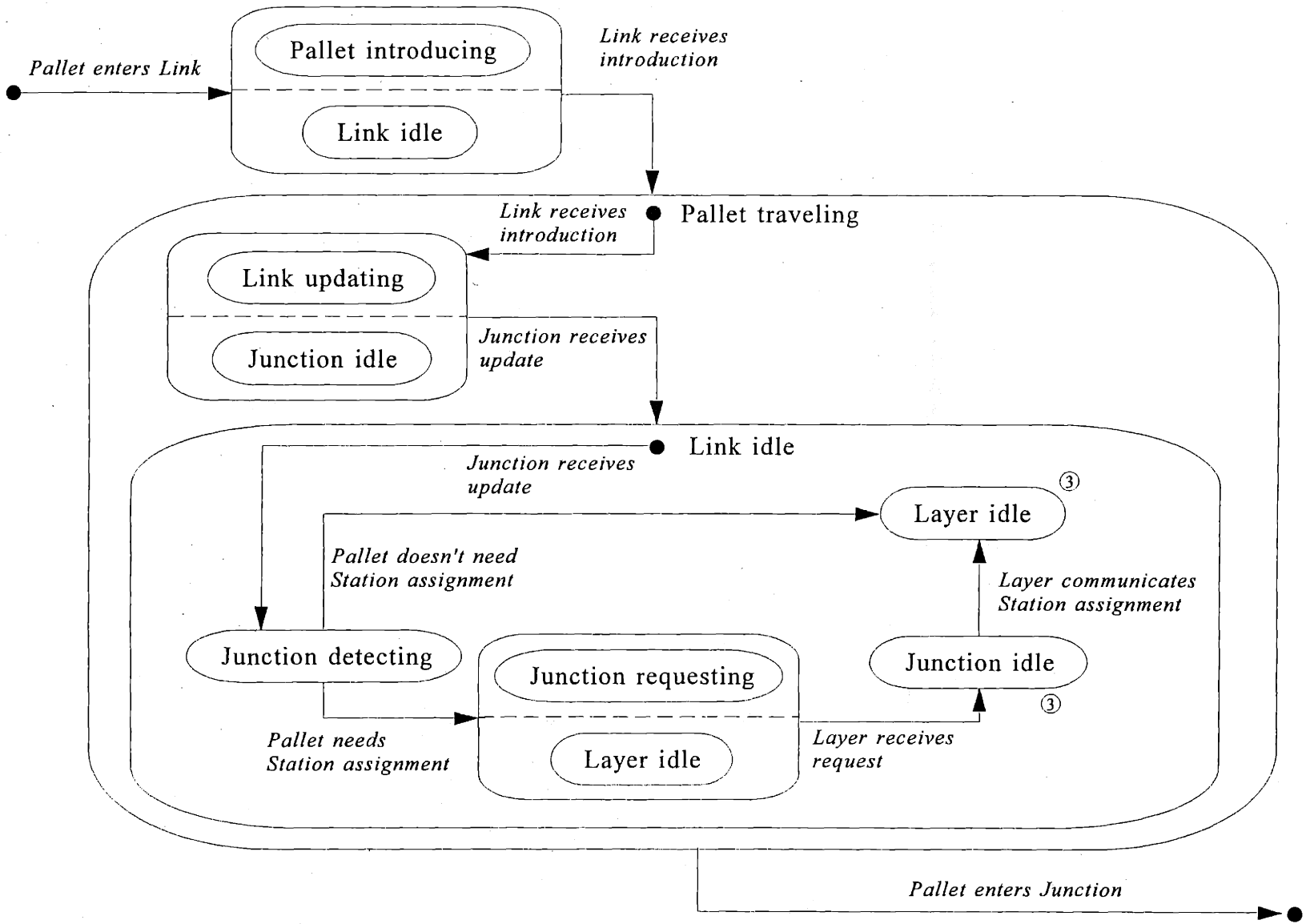
Architecture A, Action Scenario #3: Dynamic Model, Sheet 2



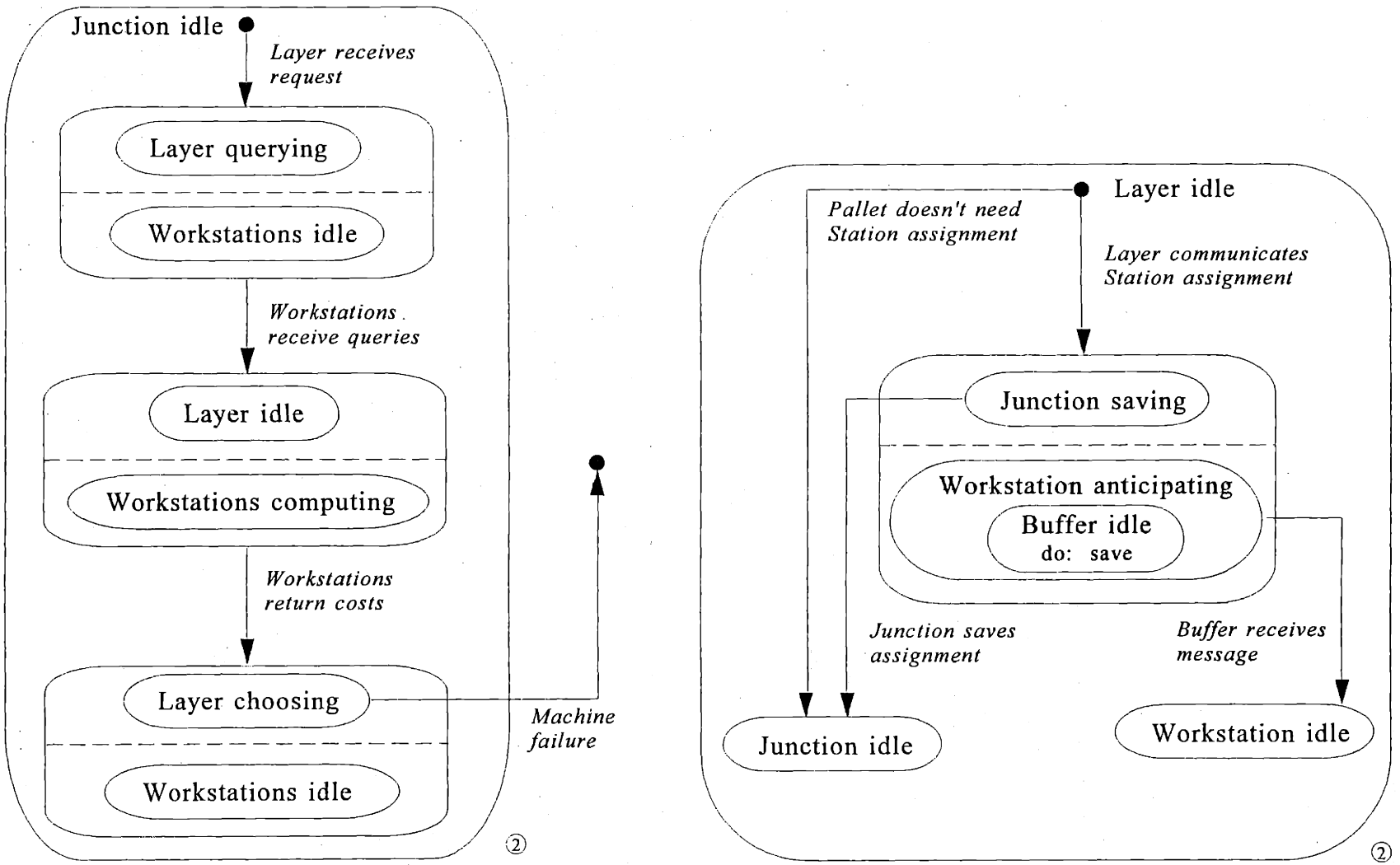
Architecture A, Action Scenario #4: Object Model



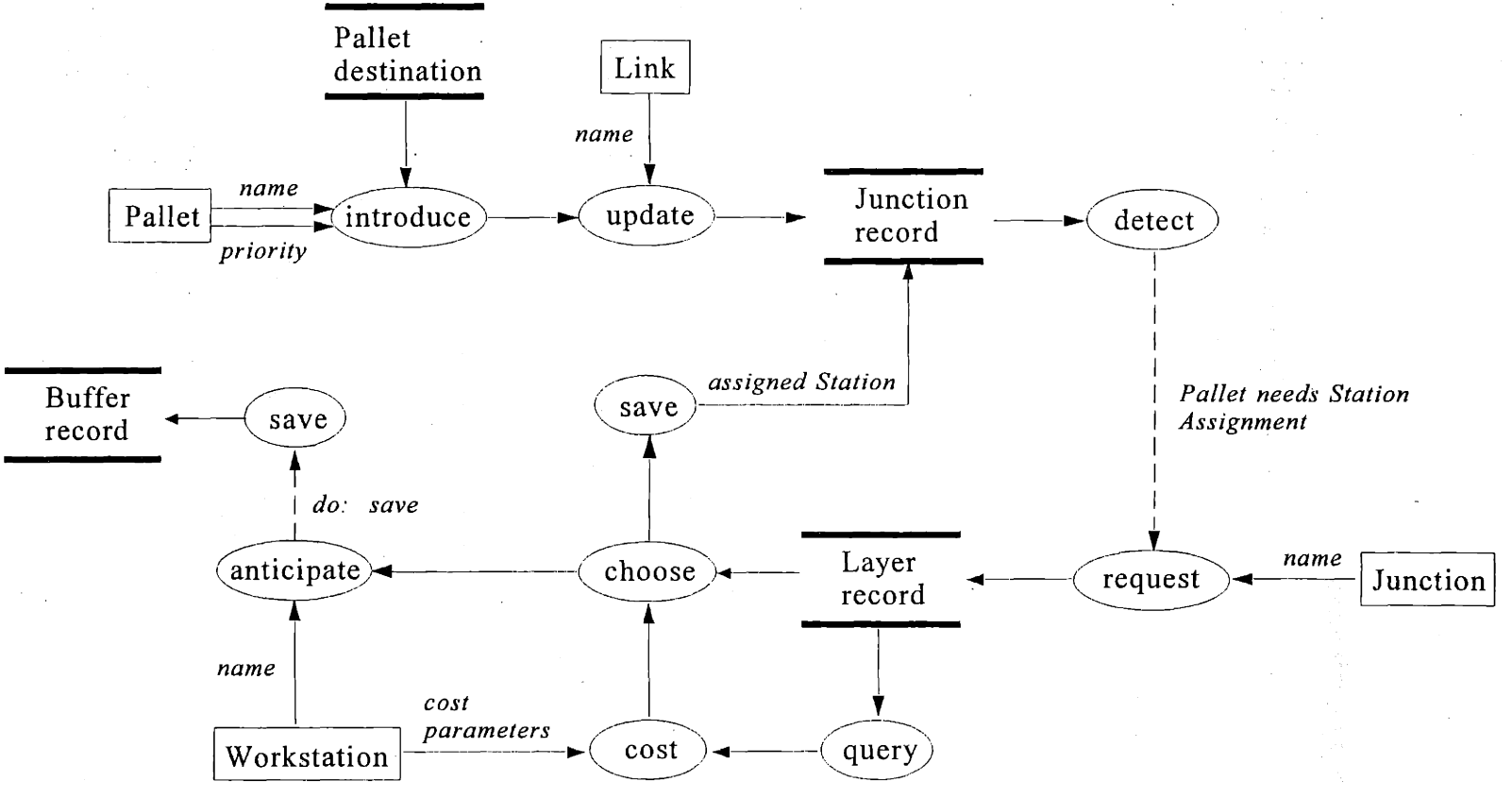
Architecture A, Action Scenario #4: Dynamic Model, Sheet 1



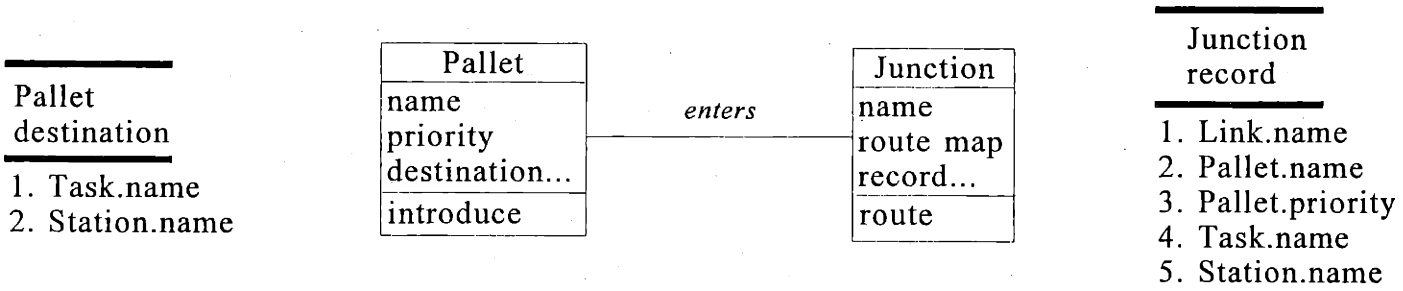
Architecture A, Action Scenario #4: Dynamic Model, Sheet 2



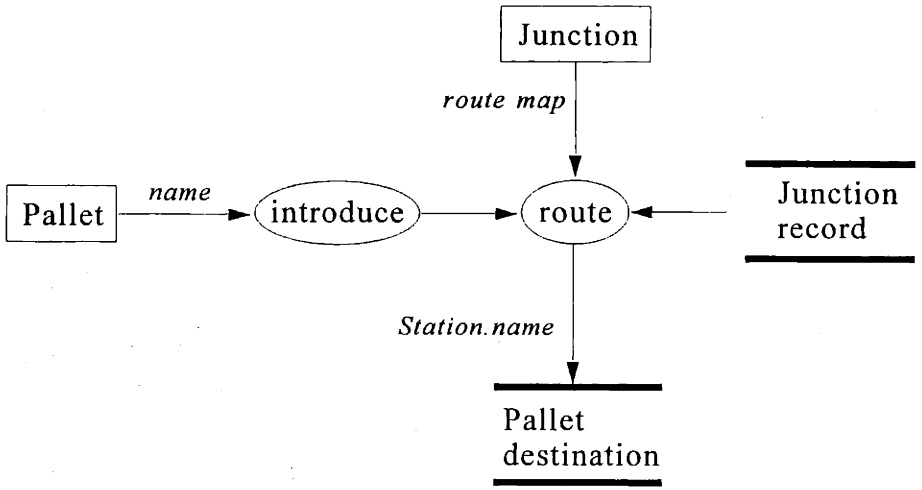
Architecture A, Action Scenario #4: Dynamic Model, Sheet 3



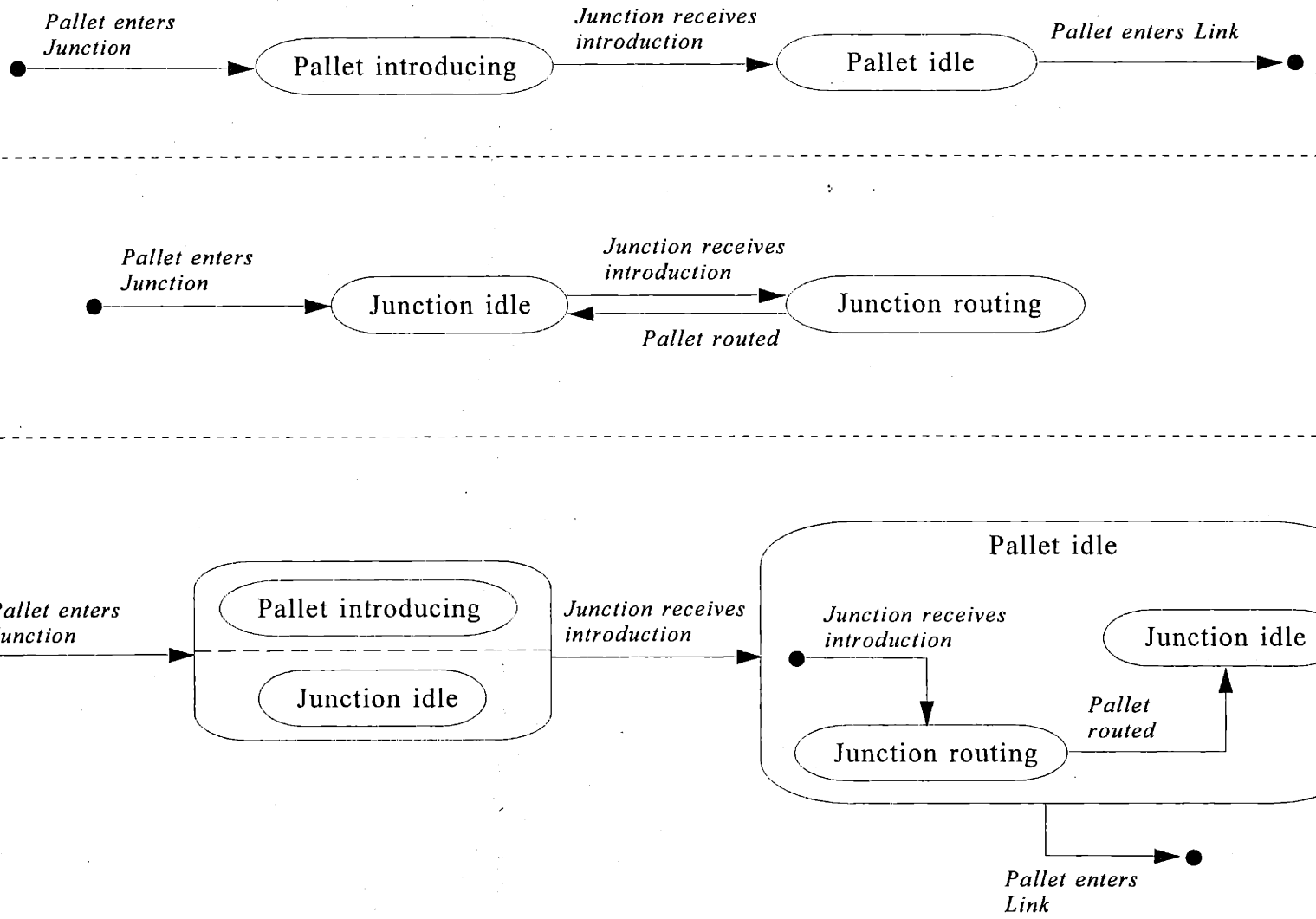
Architecture A, Action Scenario #4: Functional Model



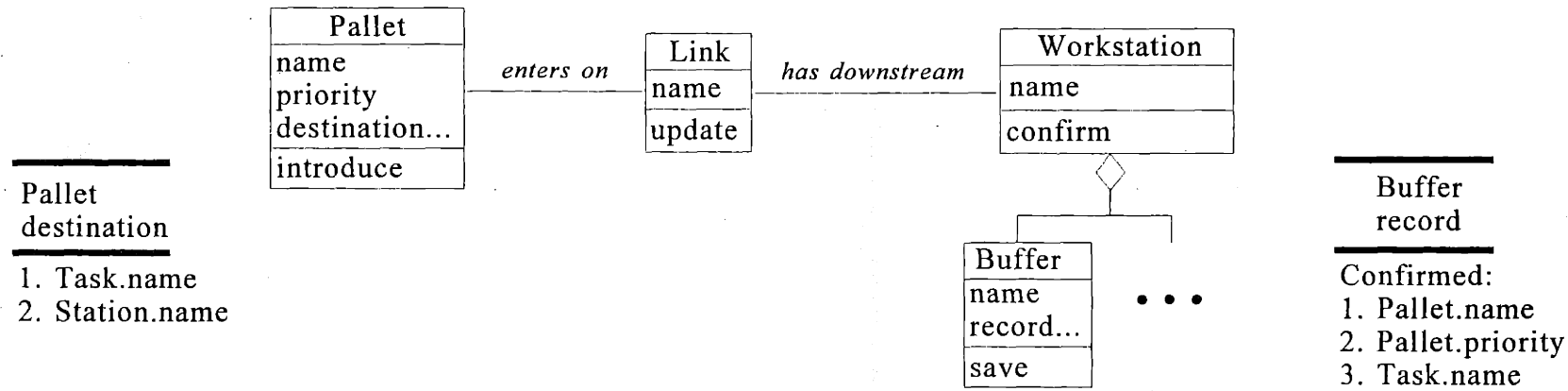
Architecture A, Action Scenario #5: Object Model



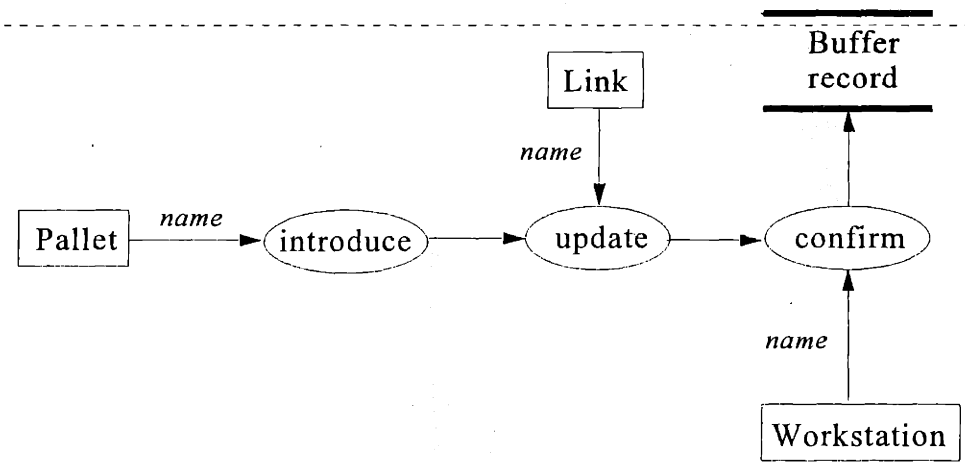
Architecture A, Action Scenario #5: Functional Model



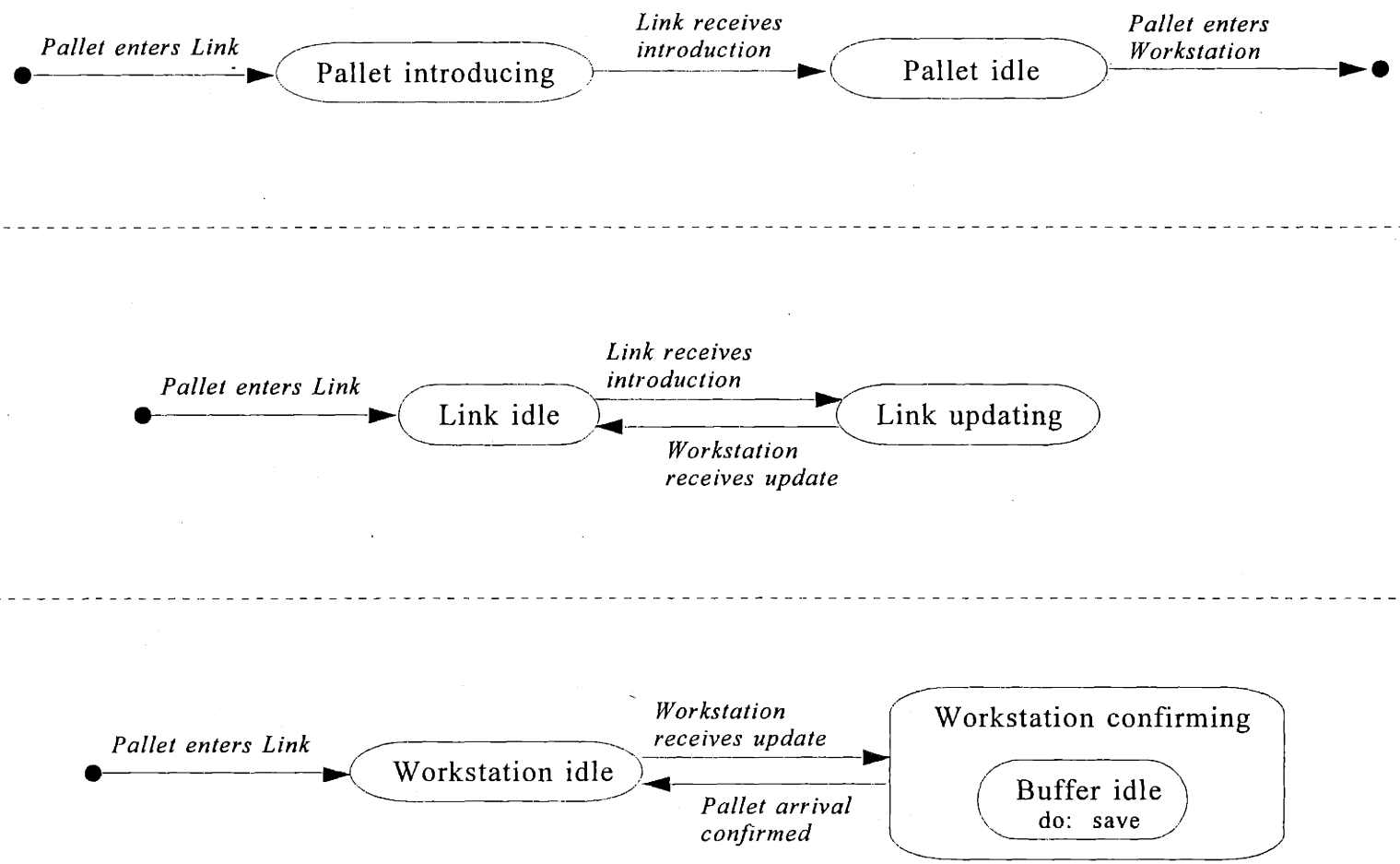
Architecture A, Action Scenario #5: Dynamic Model



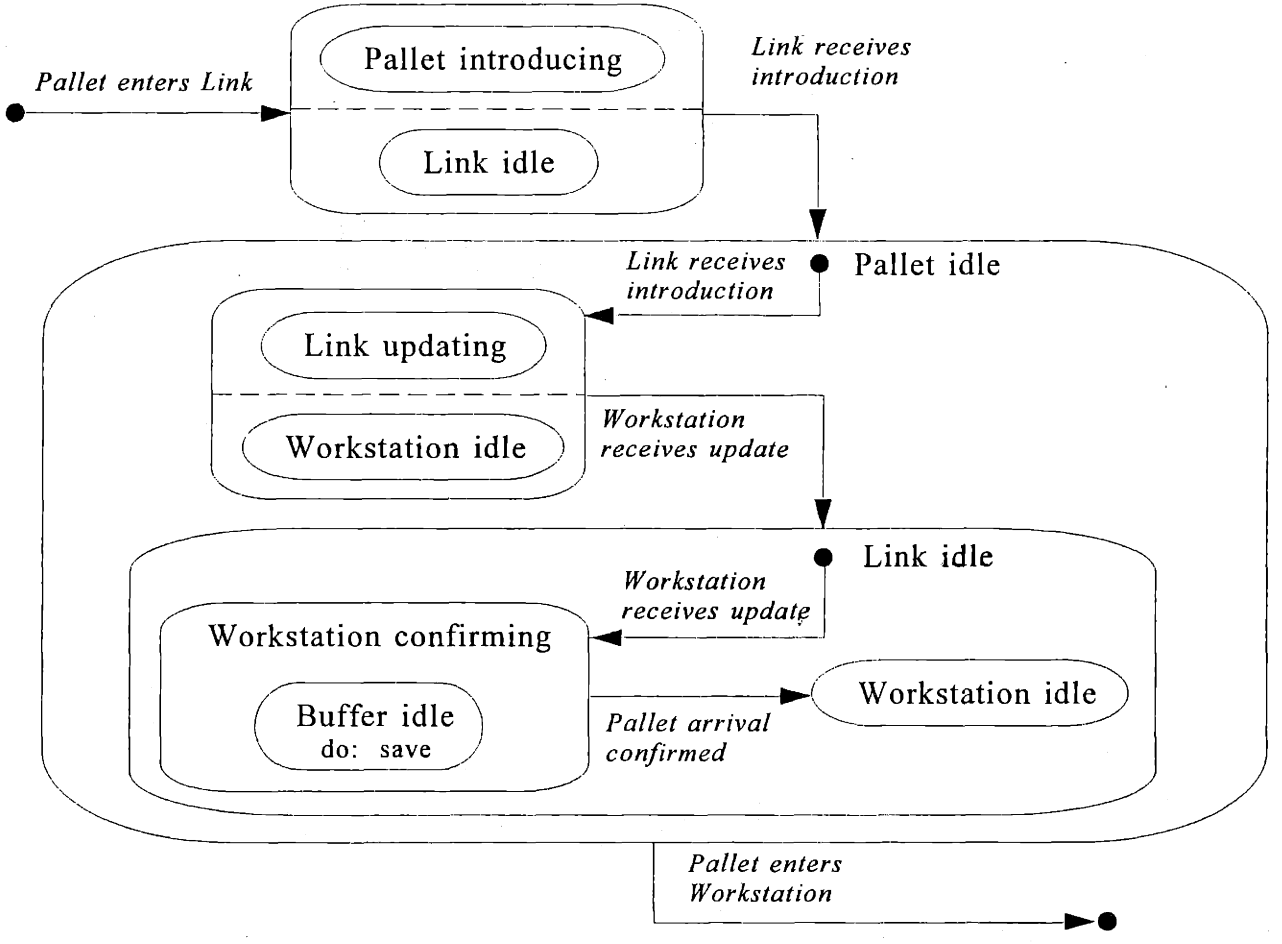
Architecture A, Action Scenario #6: Object Model



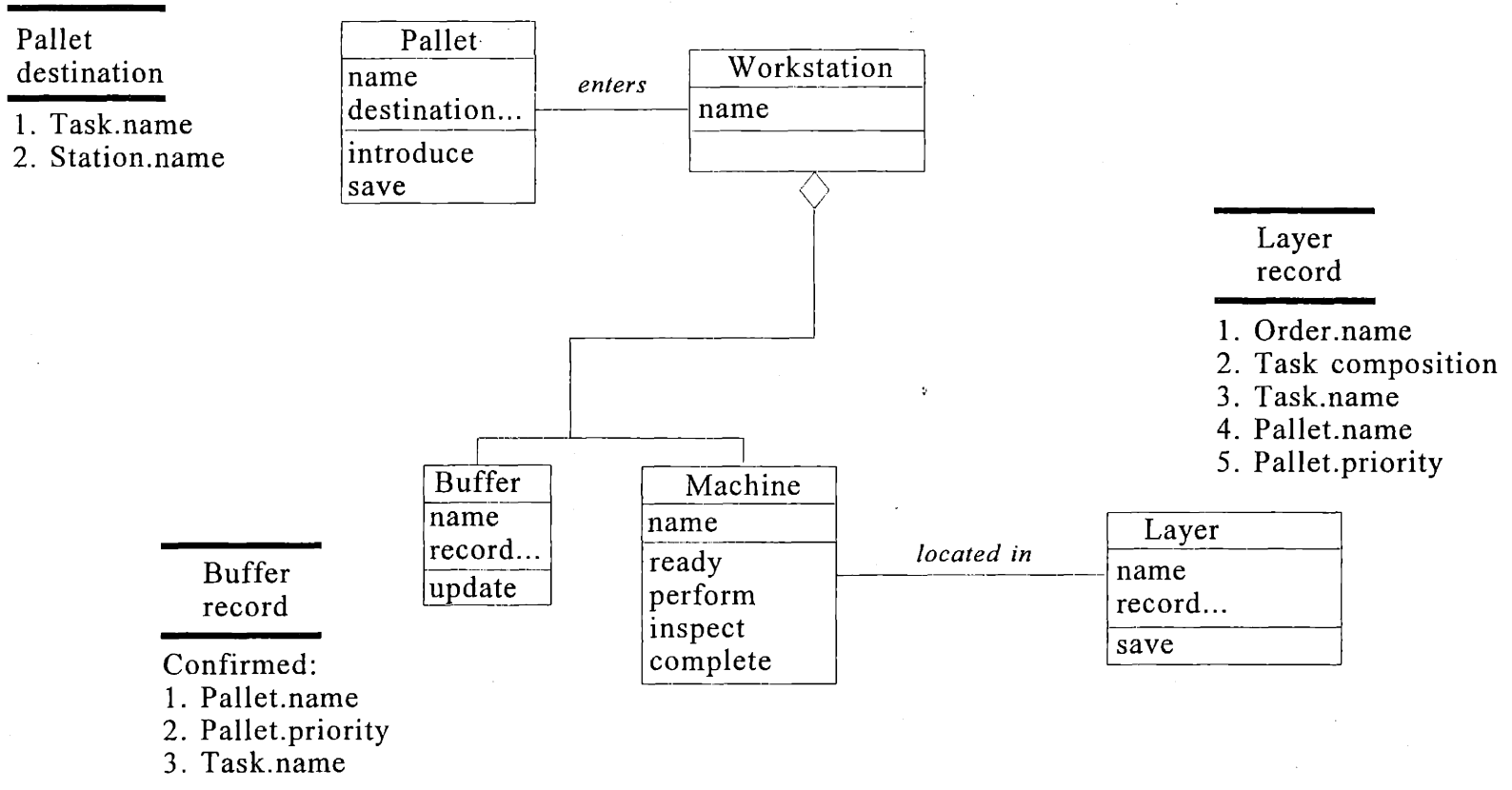
Architecture A, Action Scenario #6: Functional Model



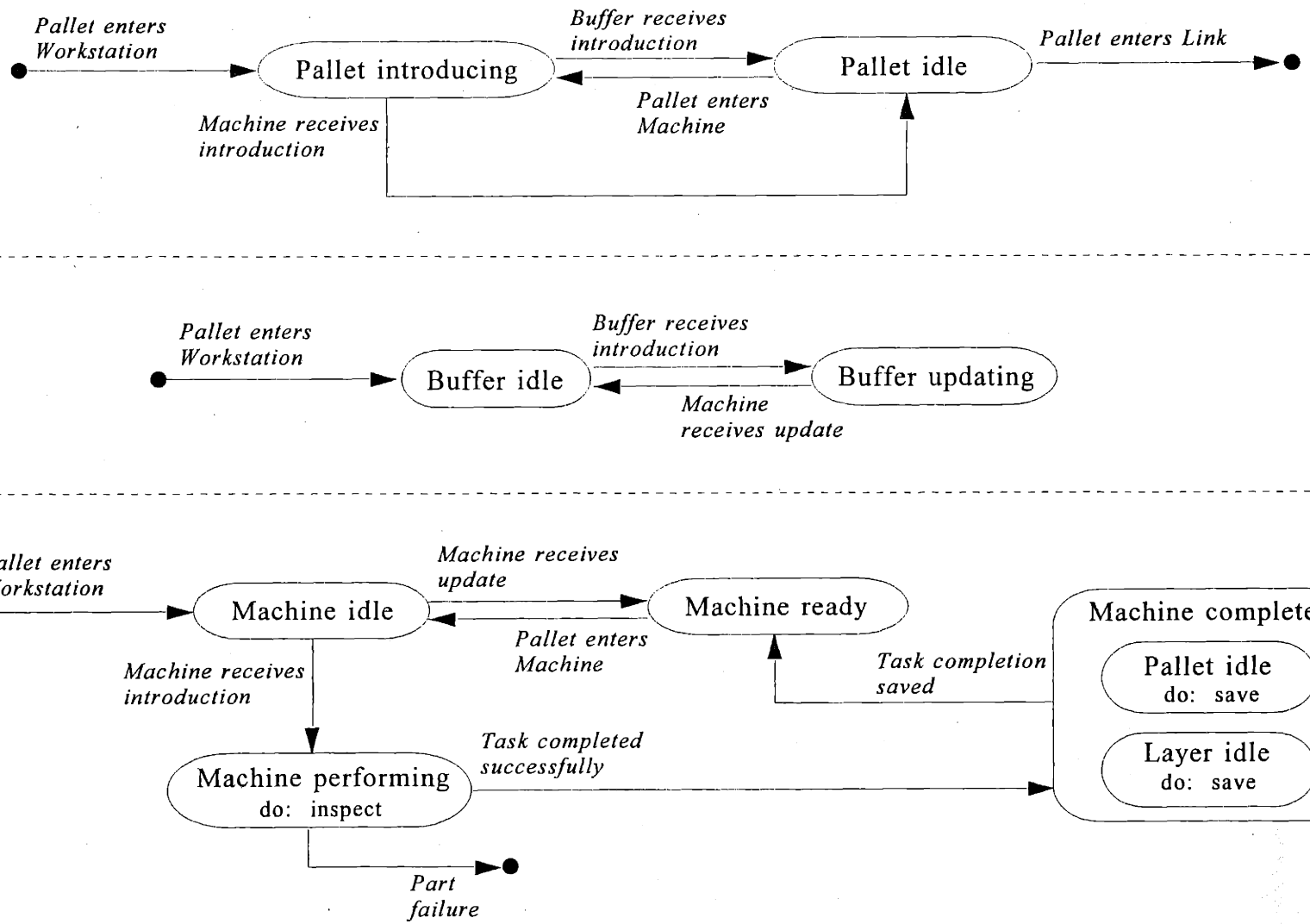
Architecture A, Action Scenario #6: Dynamic Model, Sheet 1



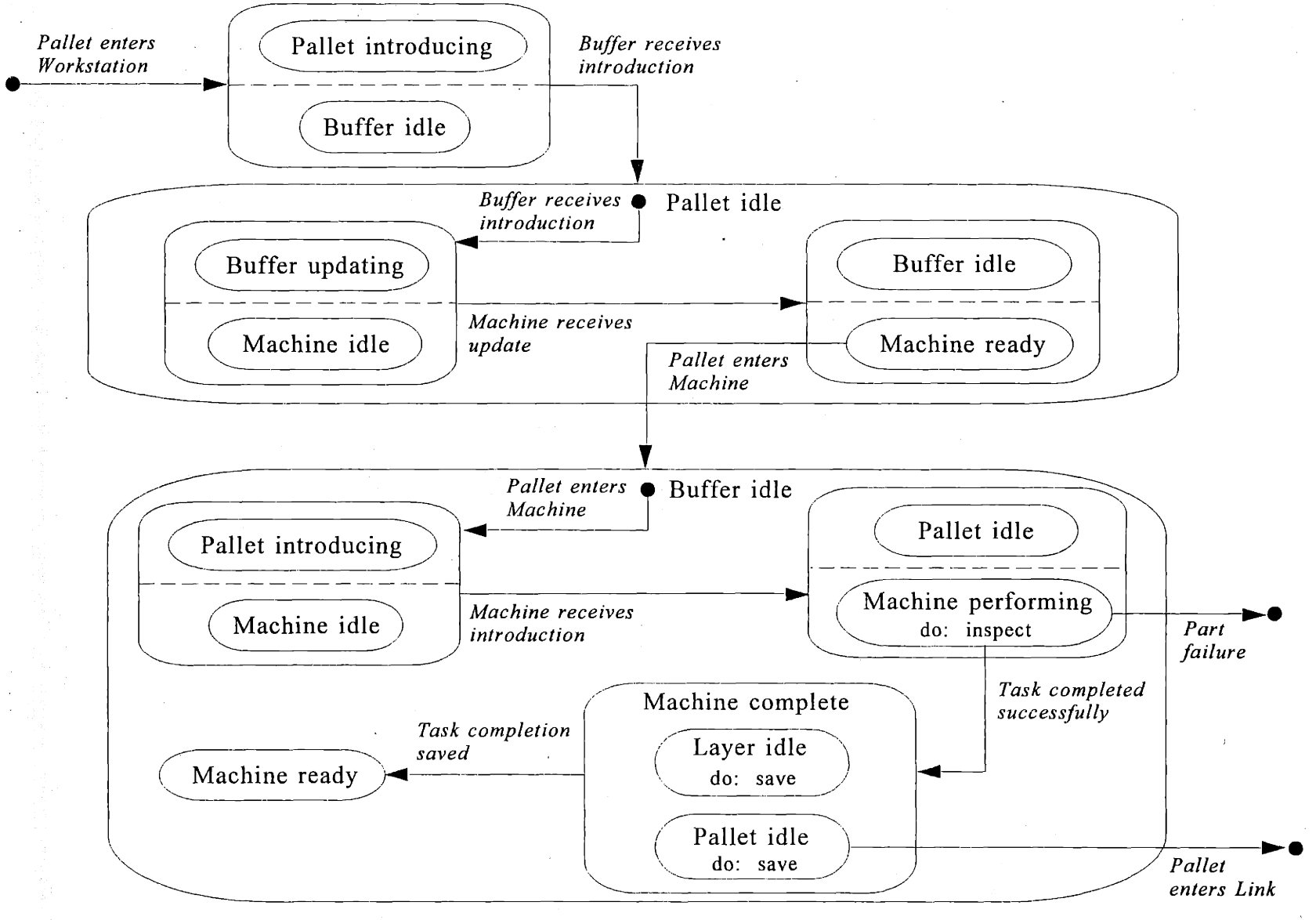
Architecture A, Action Scenario #6: Dynamic Model, Sheet 2



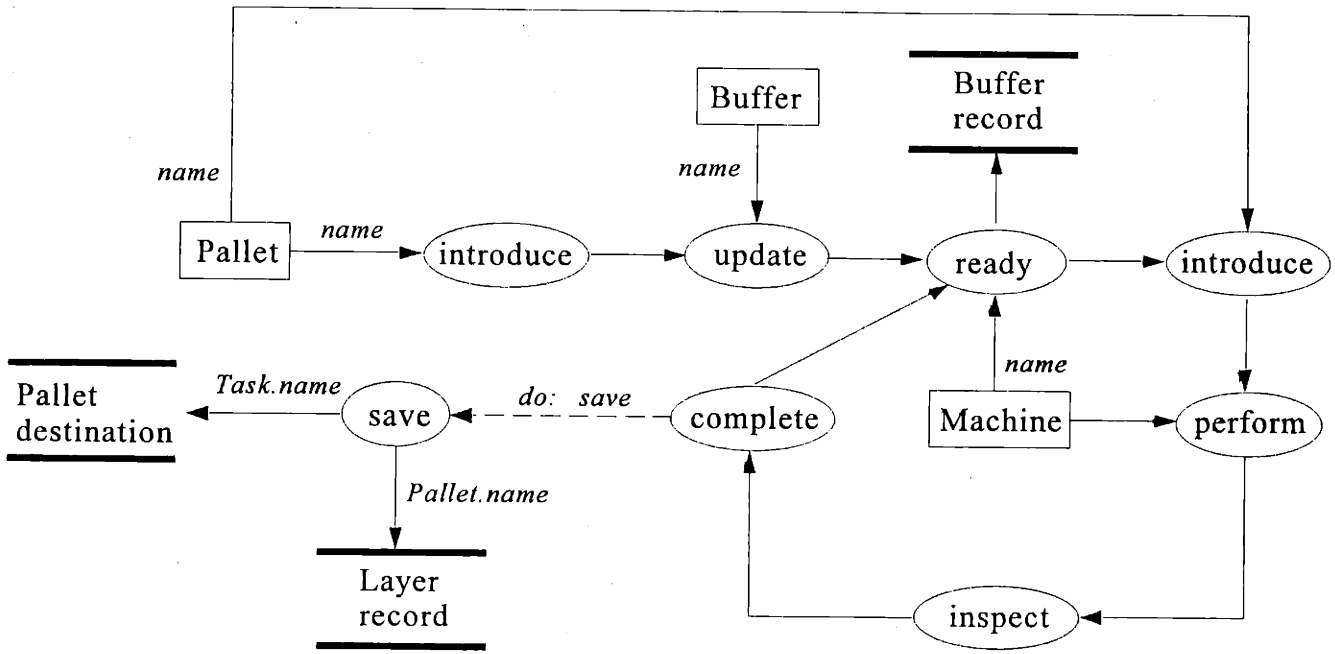
Architecture A, Action Scenario #7: Object Model



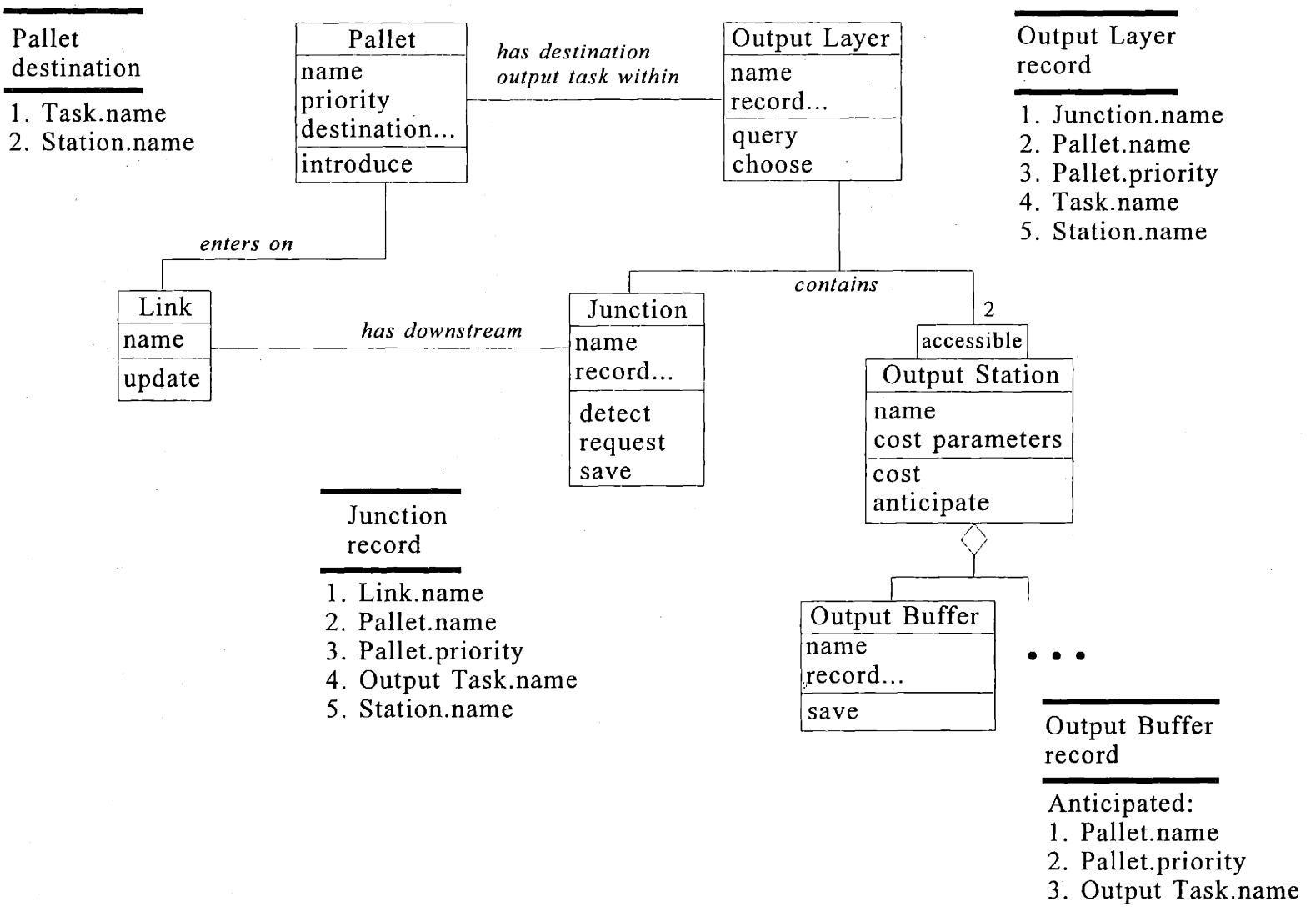
Architecture A, Action Scenario #7: Dynamic Model, Sheet 1



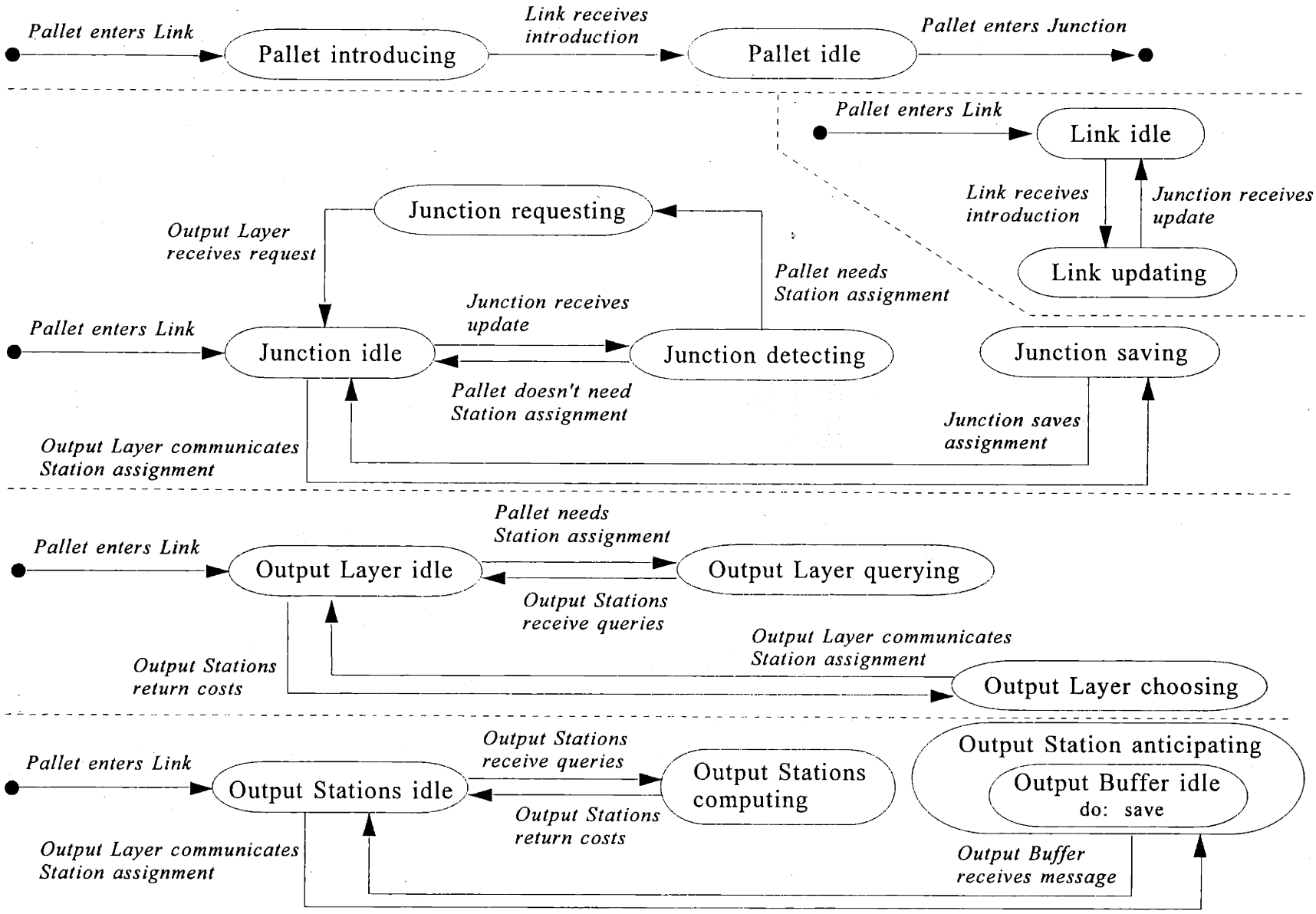
Architecture A, Action Scenario #7: Dynamic Model, Sheet 2



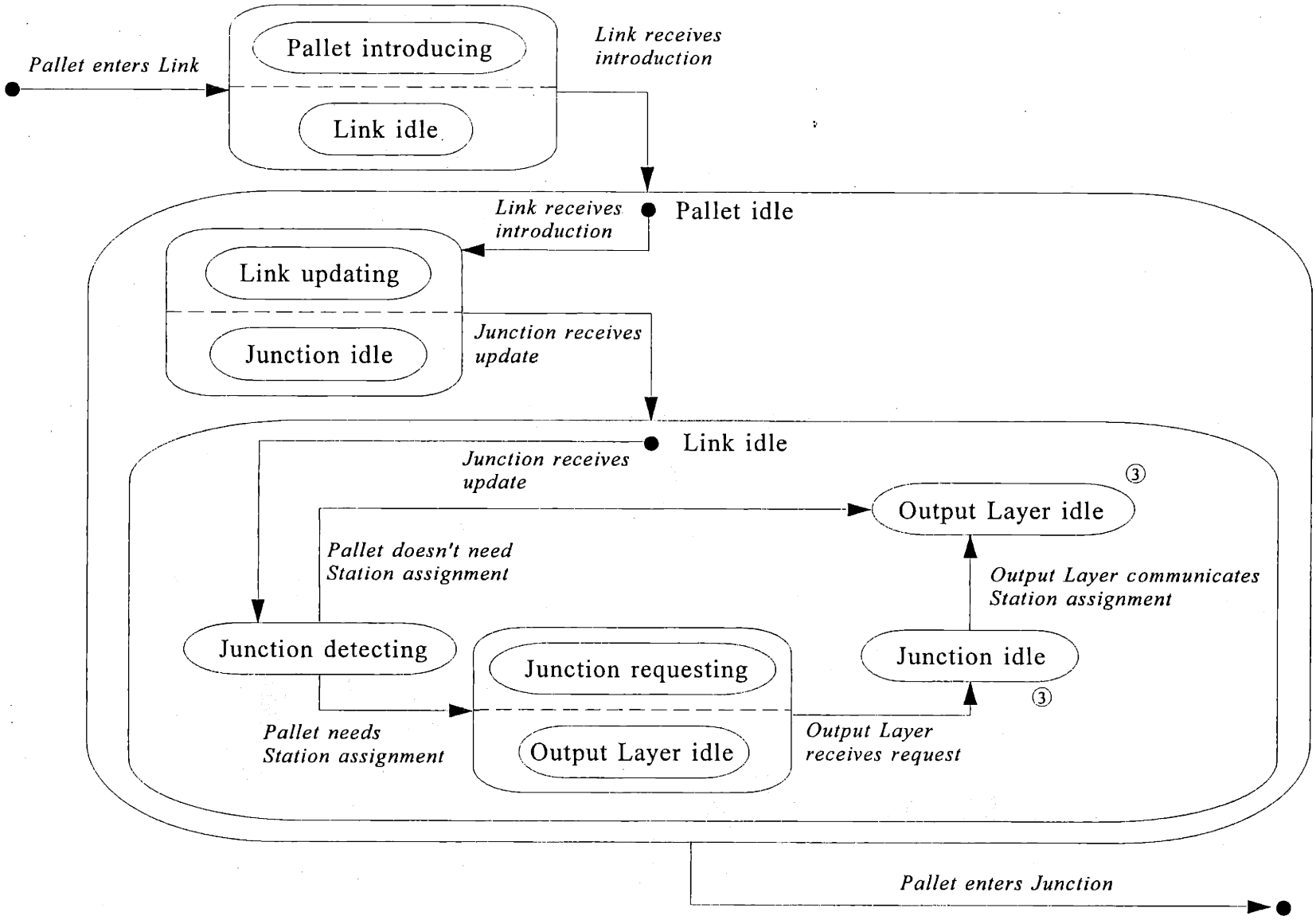
Architecture A, Action Scenario #7: Functional Model



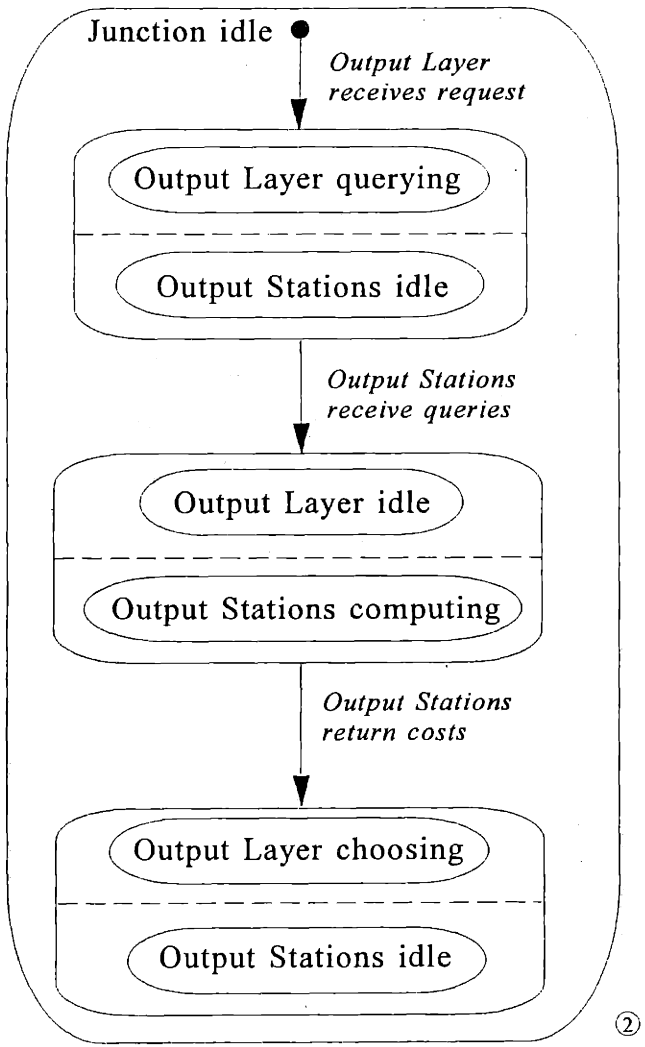
Architecture A, Action Scenario #8: Object Model

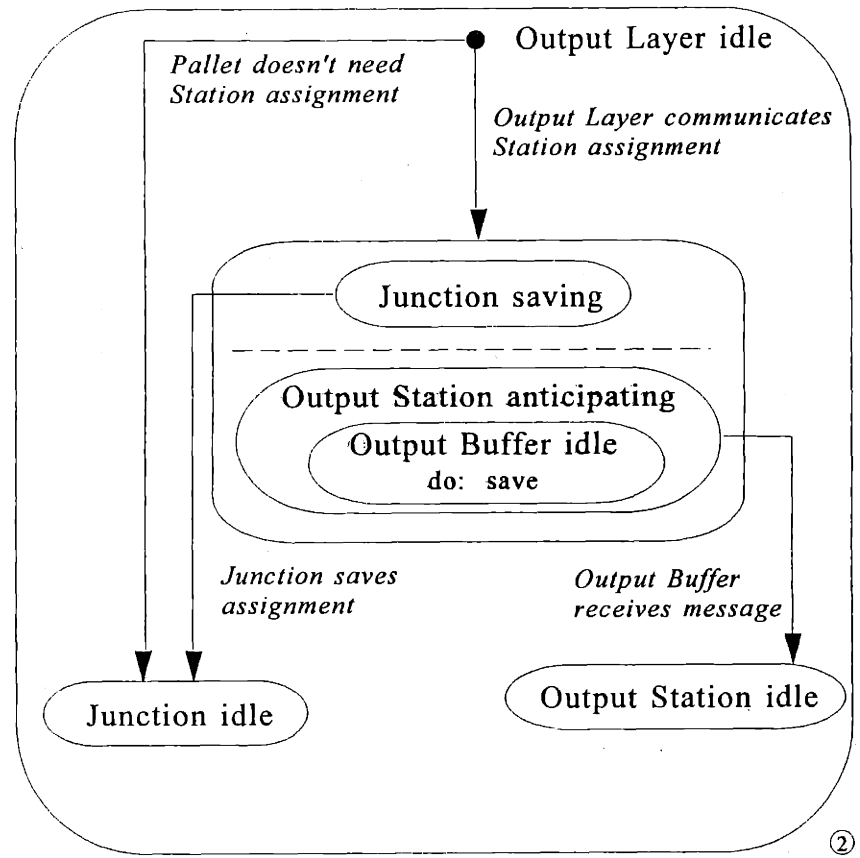


Architecture A, Action Scenario #8: Dynamic Model, Sheet 1

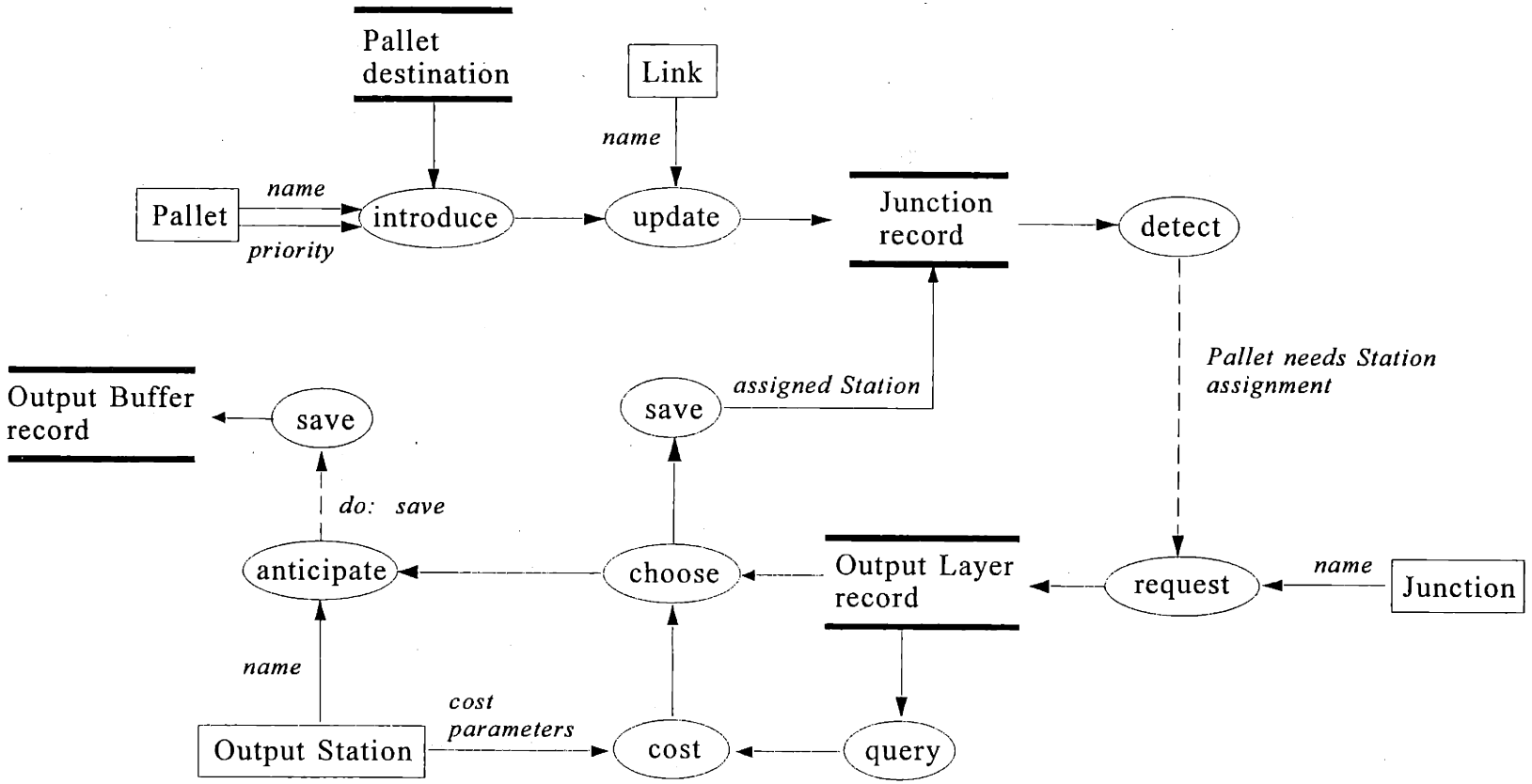


Architecture A, Action Scenario #8: Dynamic Model, Sheet 2

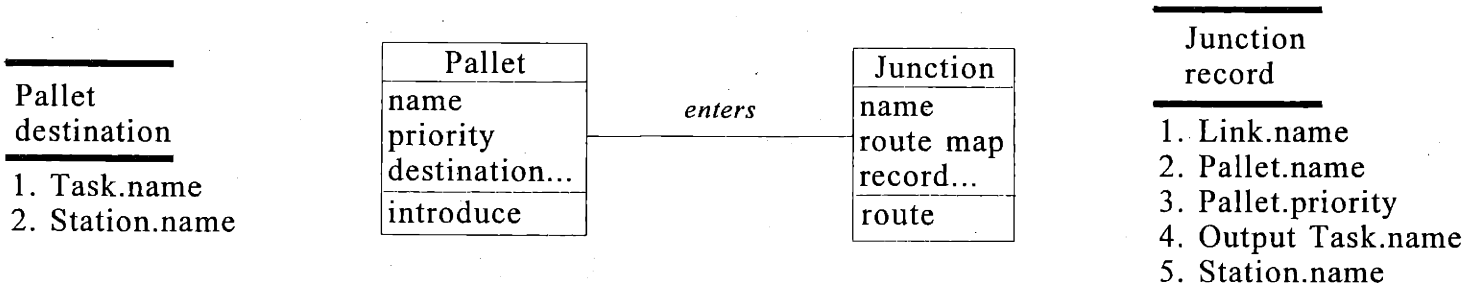




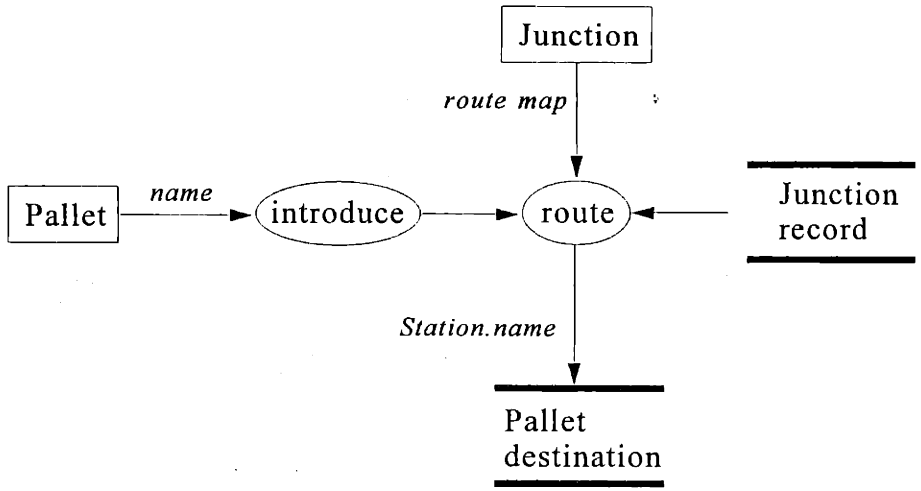
Architecture A, Action Scenario #8: Dynamic Model, Sheet 3



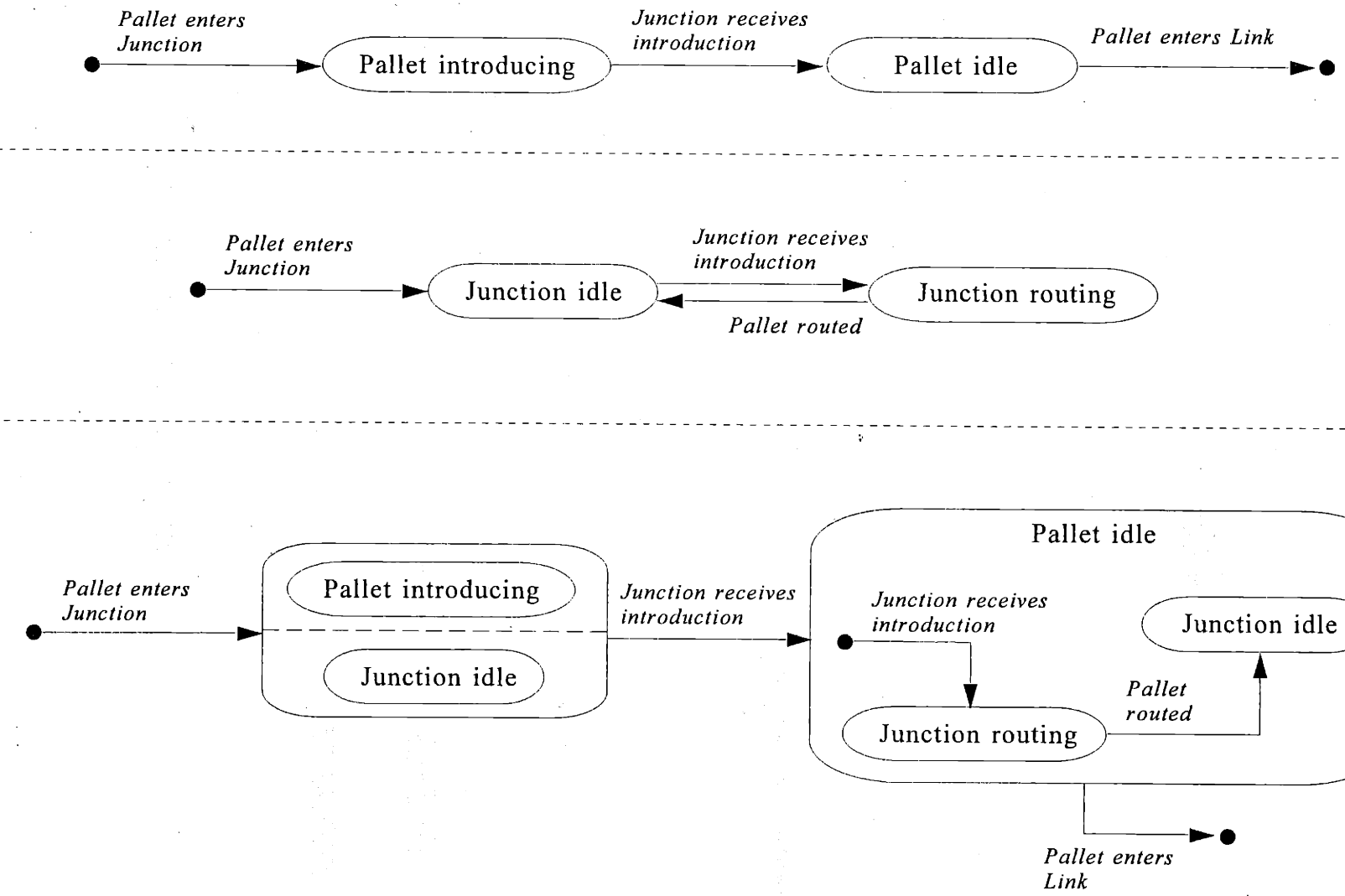
Architecture A, Action Scenario #8: Functional Model



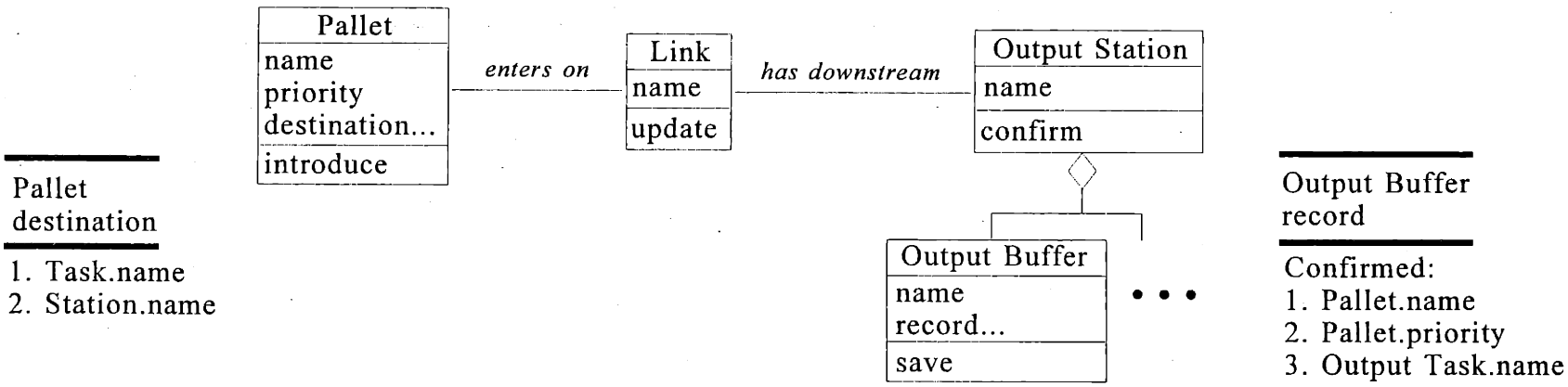
Architecture A, Action Scenario #9: Object Model



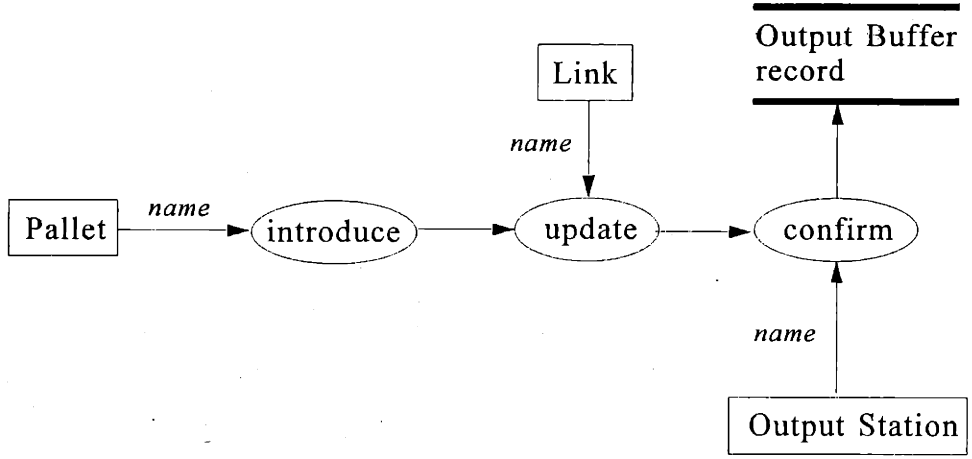
Architecture A, Action Scenario #9: Functional Model



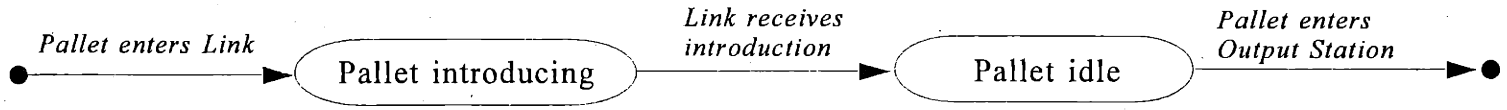
Architecture A, Action Scenario #9: Dynamic Model

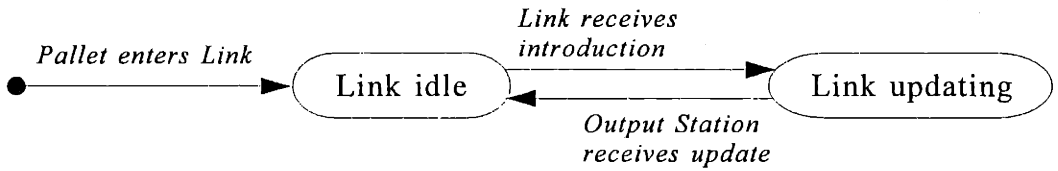


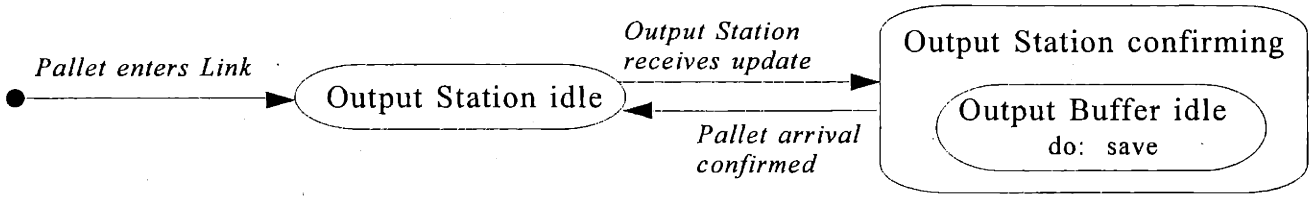
Architecture A, Action Scenario #10: Object Model



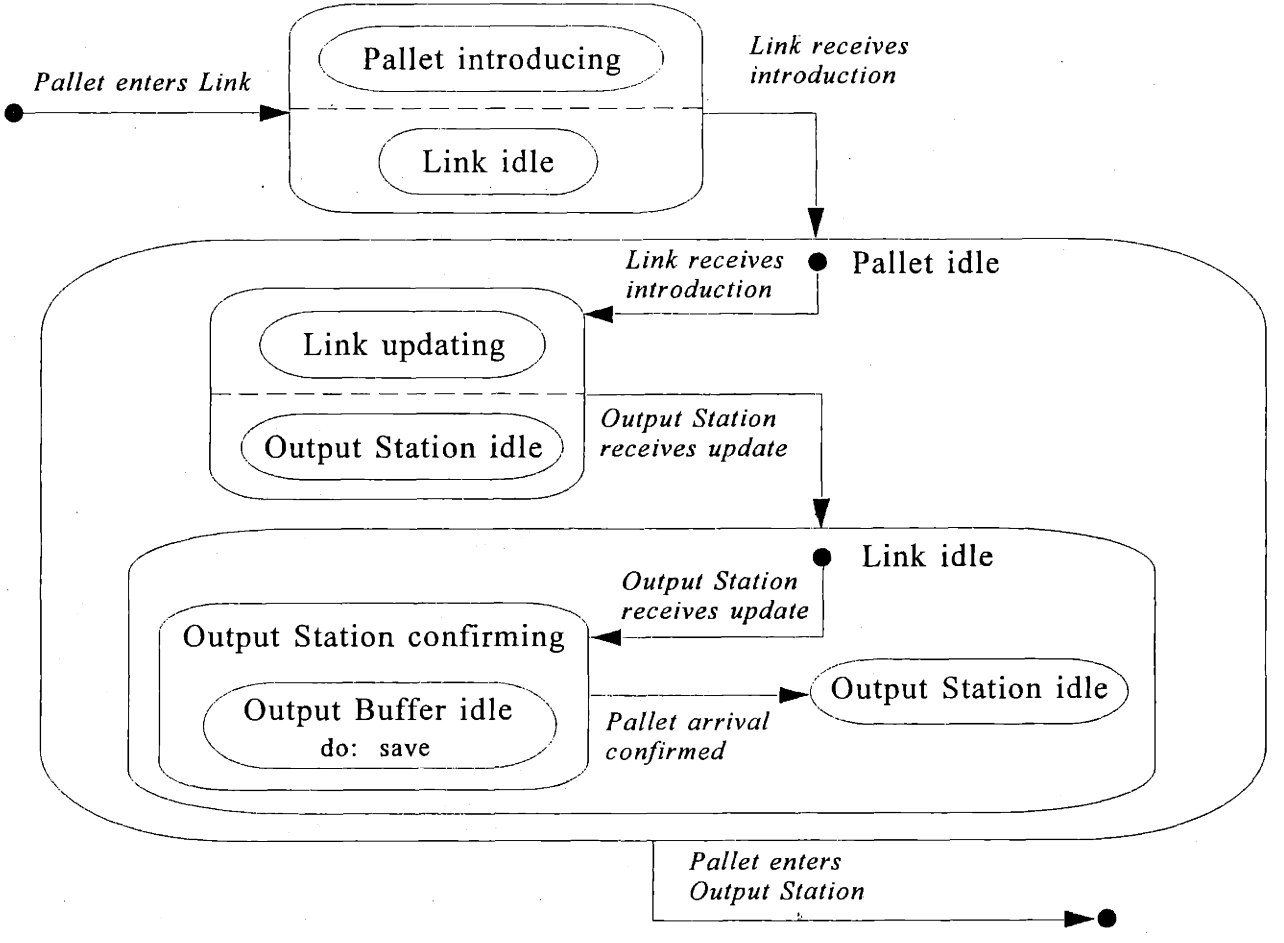
Architecture A, Action Scenario #10: Functional Model



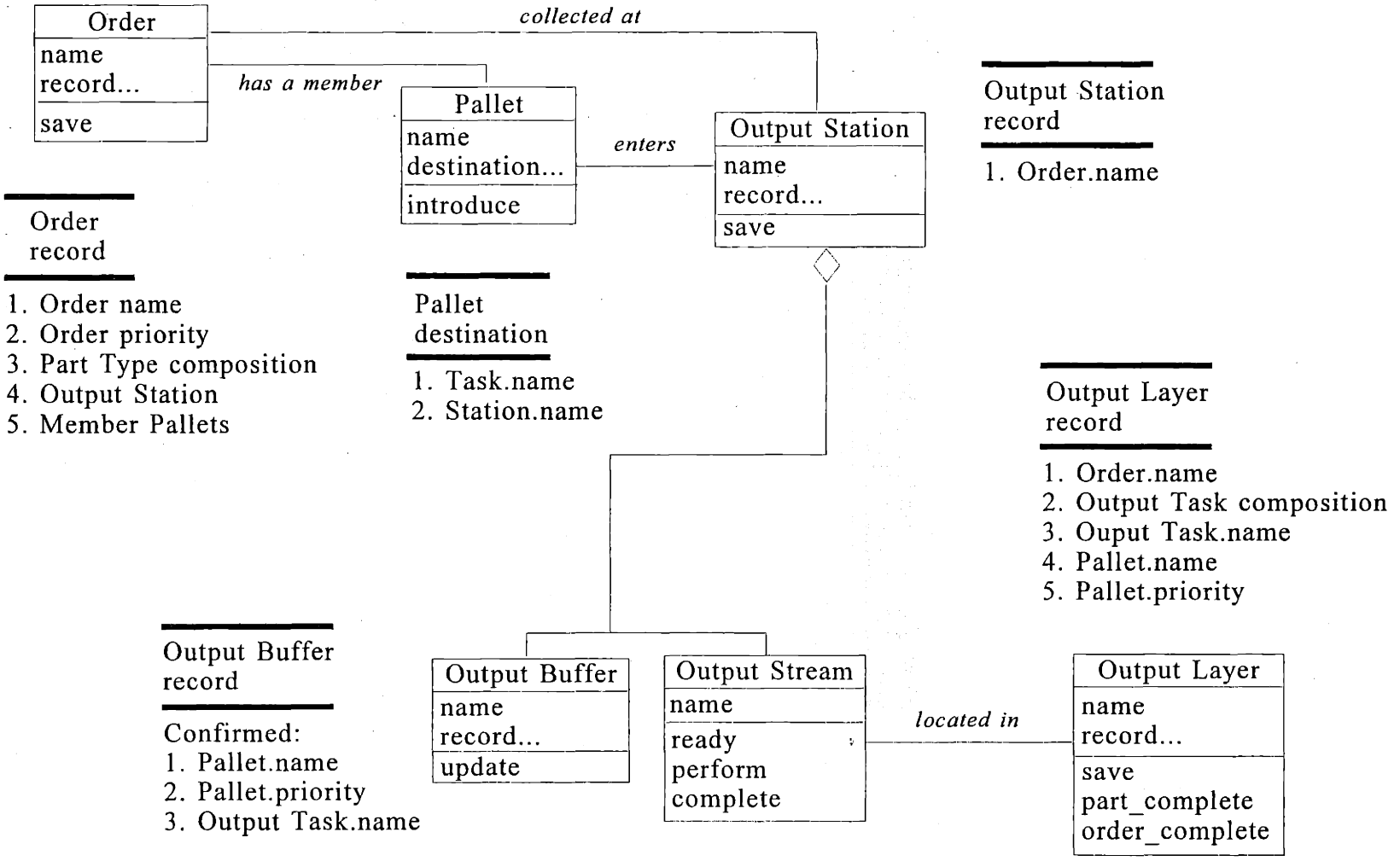




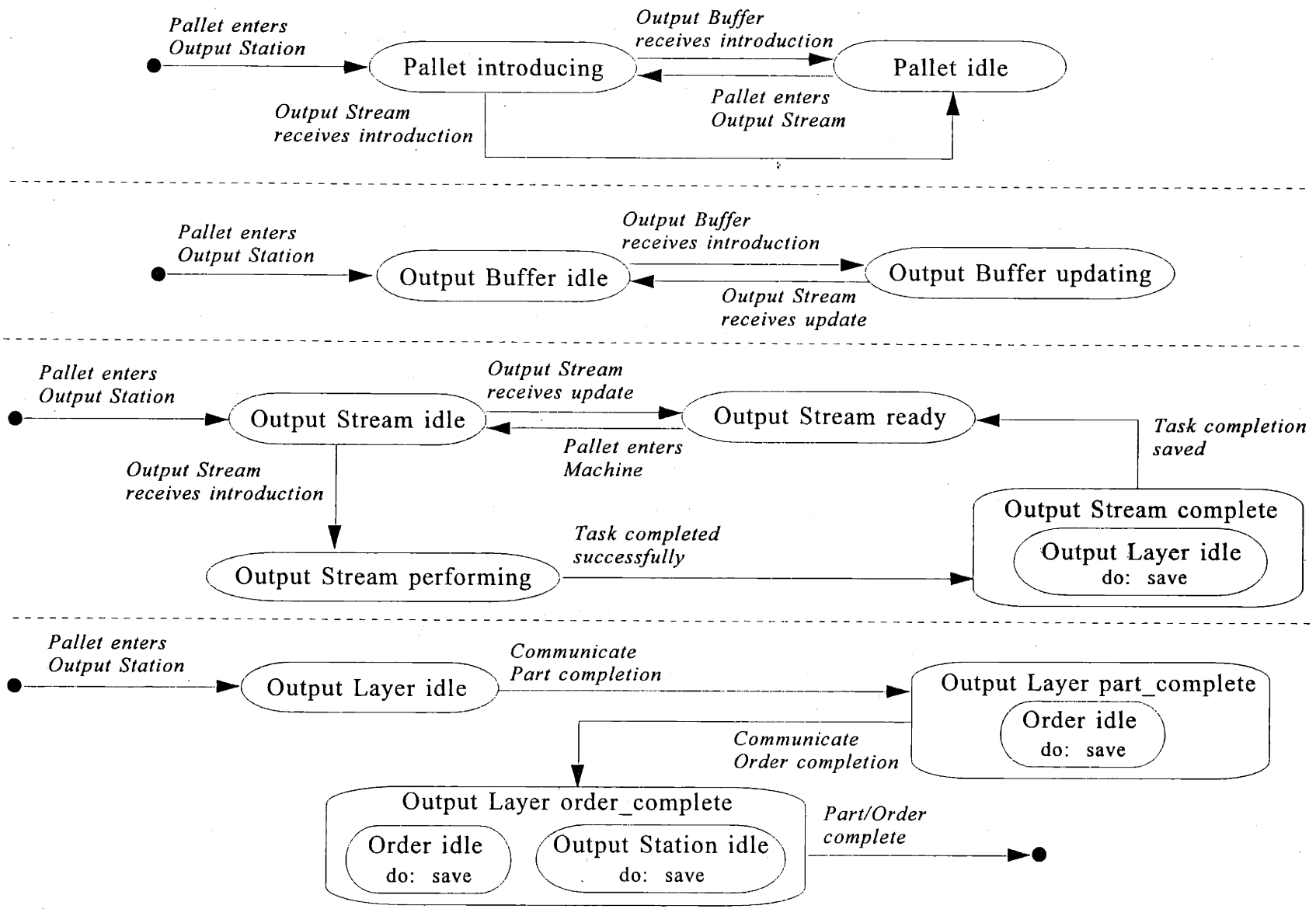
Architecture A, Action Scenario #10: Dynamic Model, Sheet 1



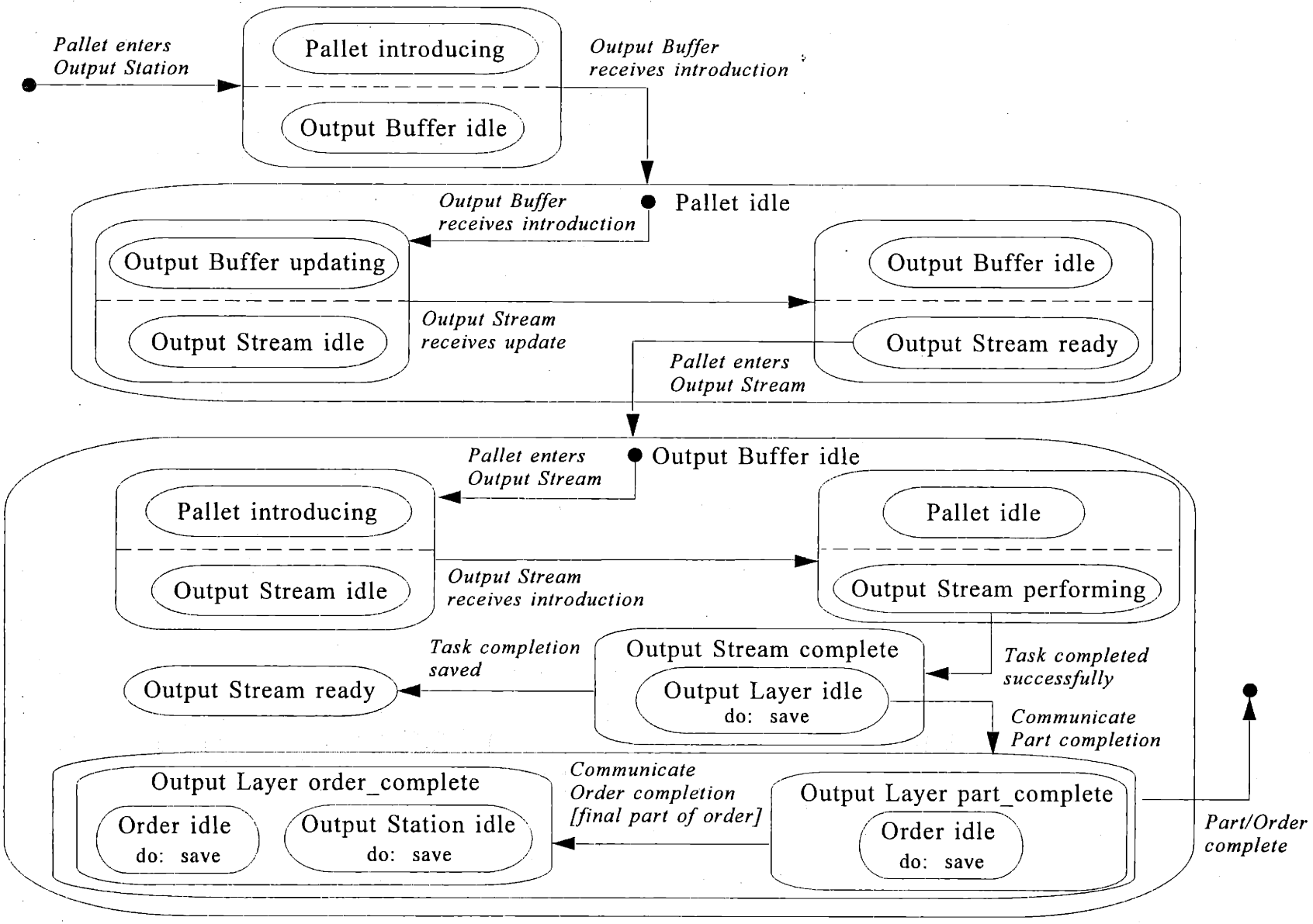
Architecture A, Action Scenario #10: Dynamic Model, Sheet 2



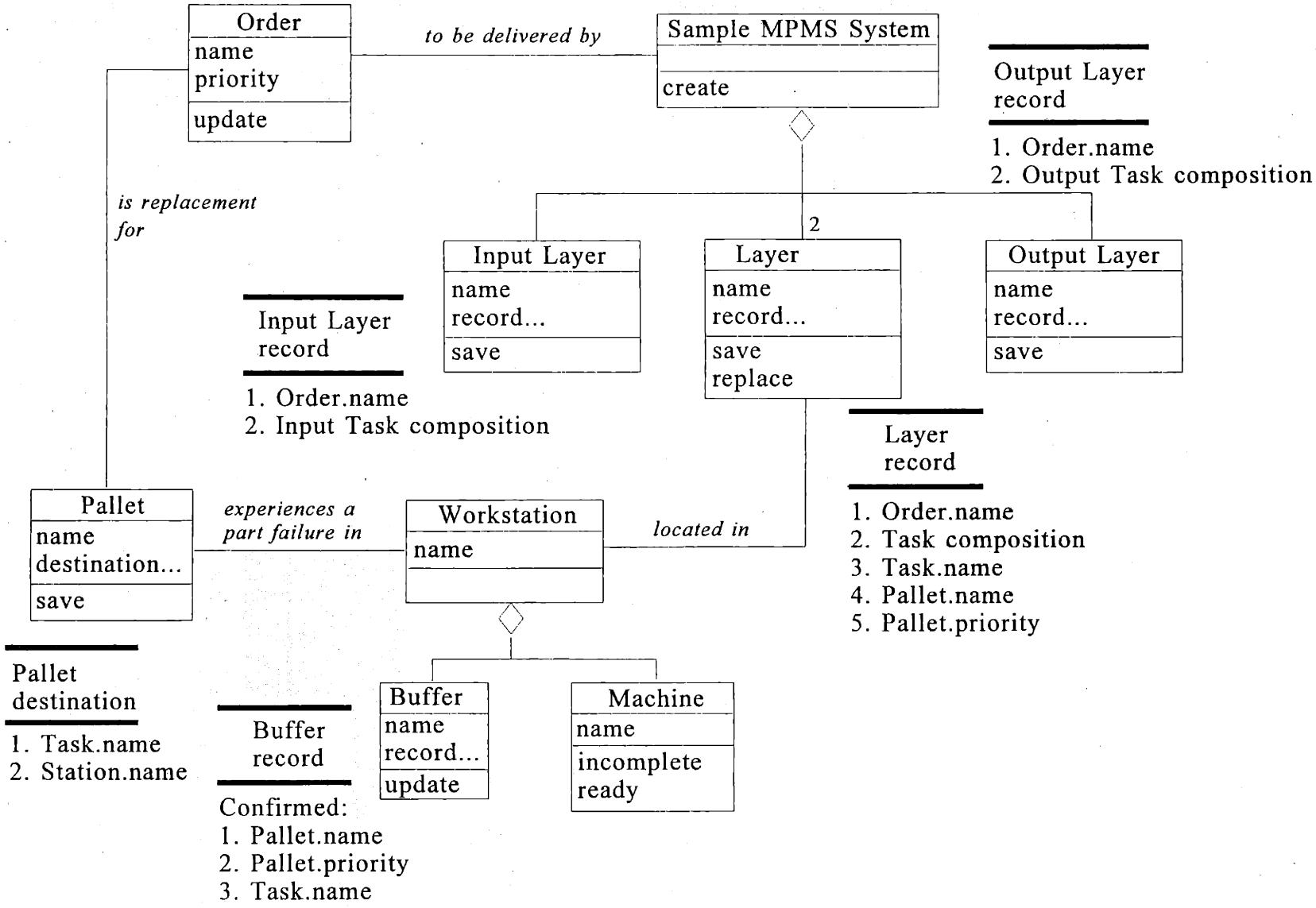
Architecture A, Action Scenario #11: Object Model



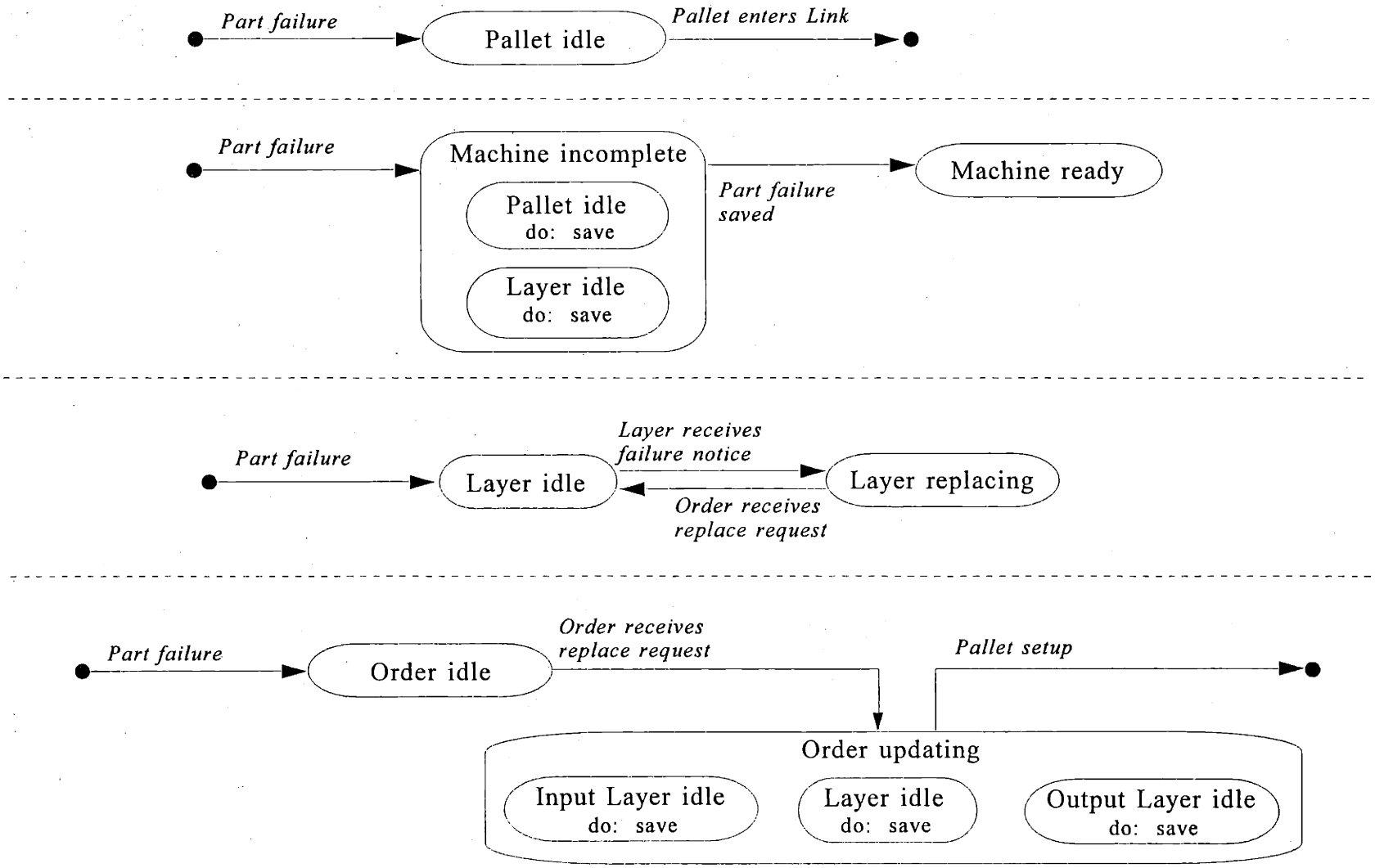
Architecture A, Action Scenario #11: Dynamic Model, Sheet 1



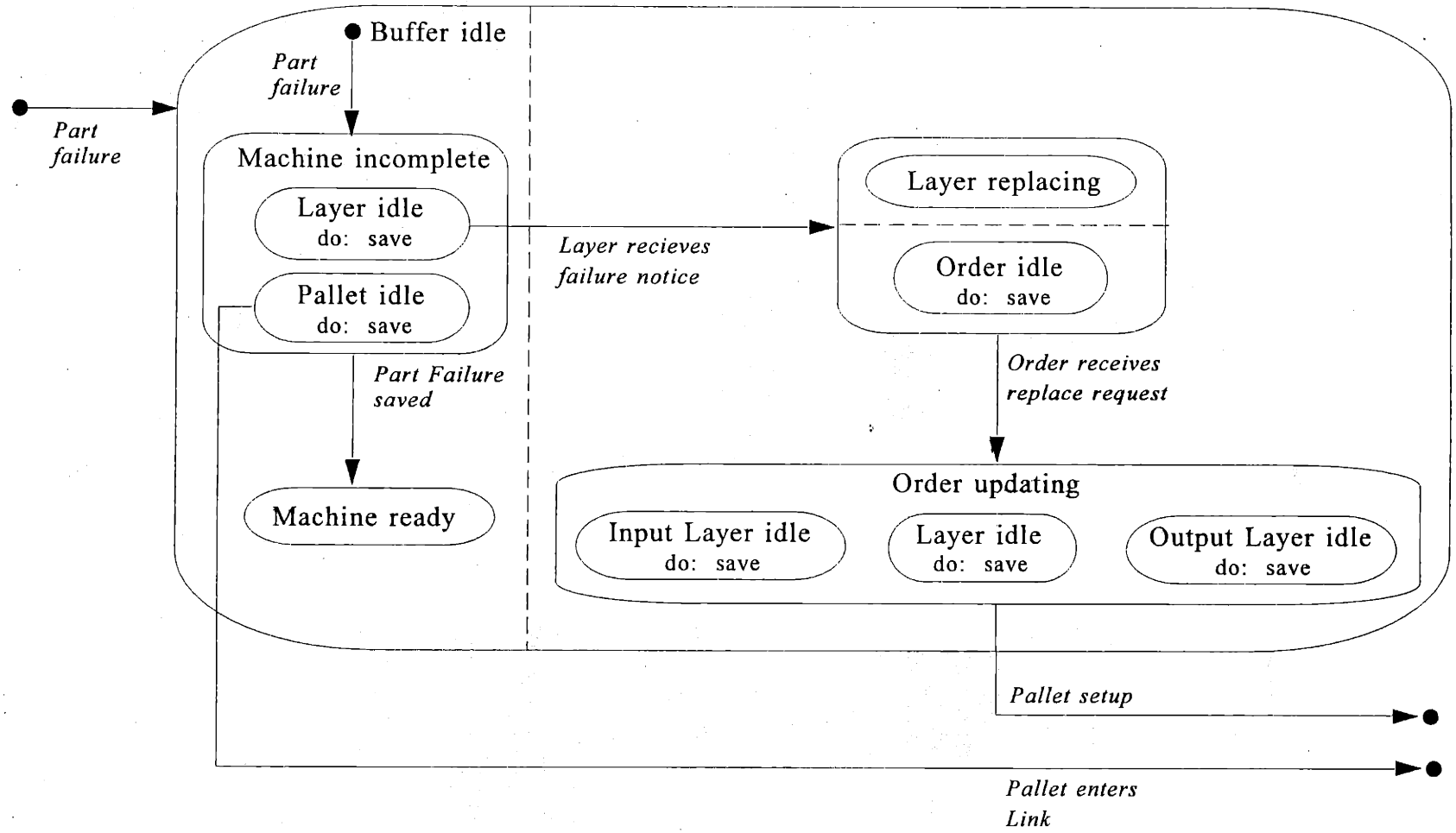
Architecture A, Action Scenario #11: Dynamic Model, Sheet 2



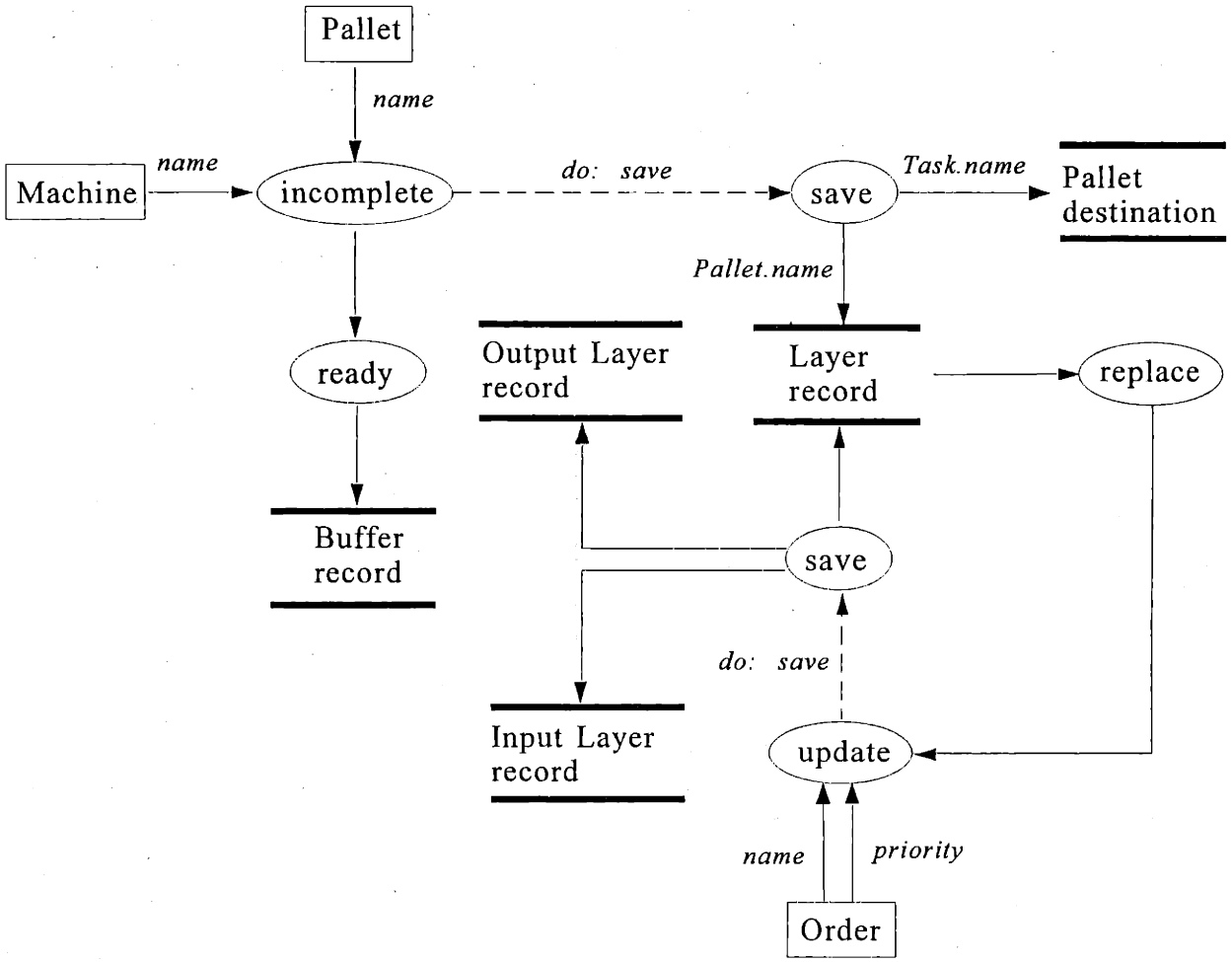
Architecture A, Action Scenario #12: Object Model



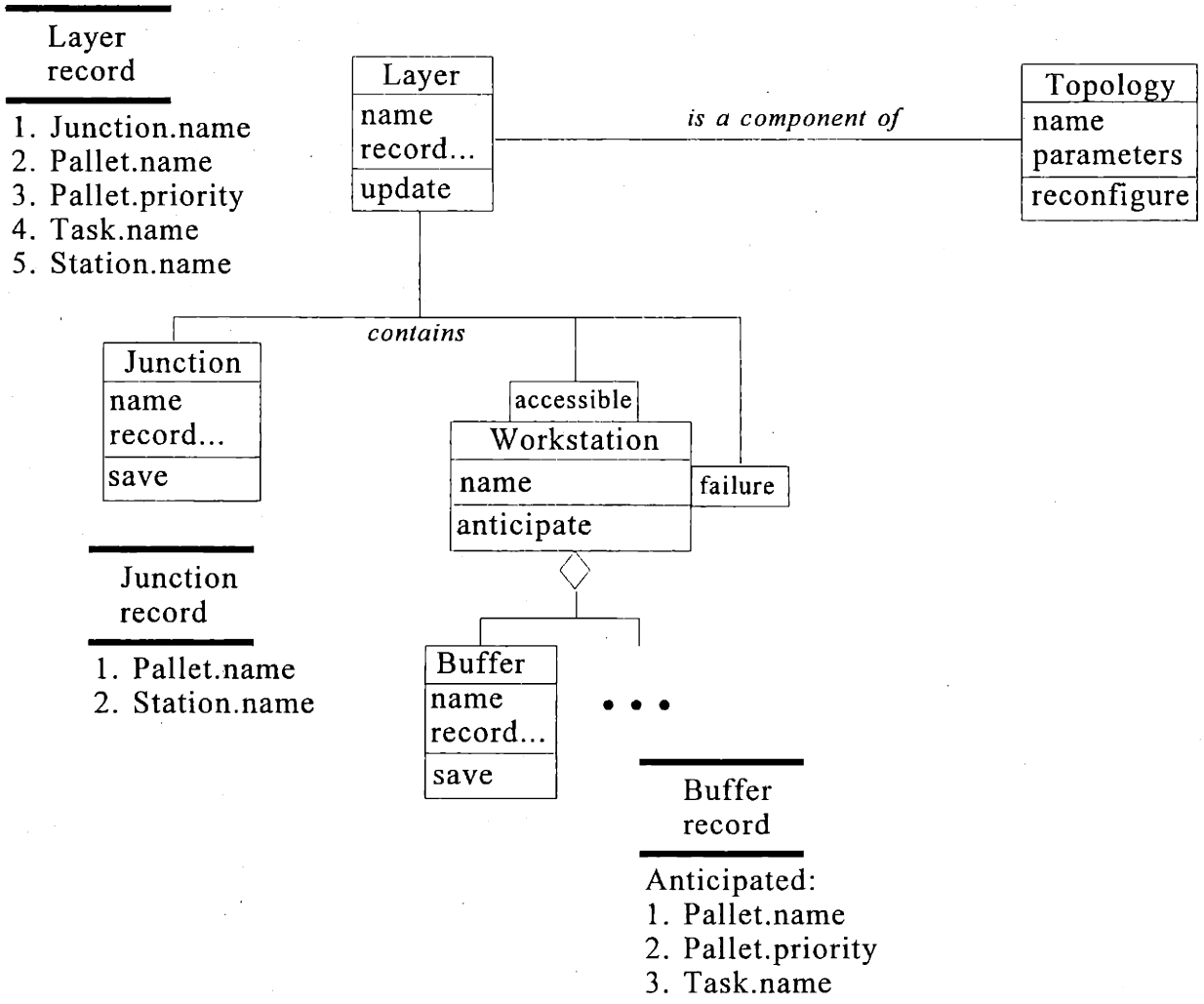
Architecture A, Action Scenario #12: Dynamic Model, Sheet 1



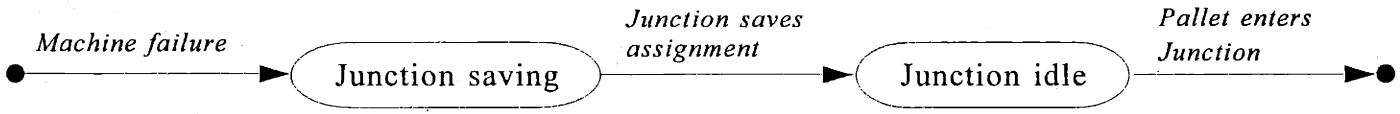
Architecture A, Action Scenario #12: Dynamic Model, Sheet 2

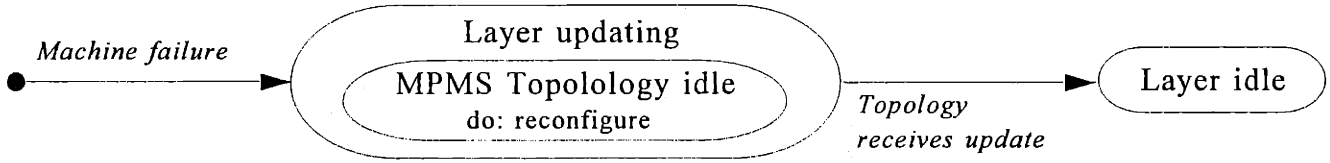


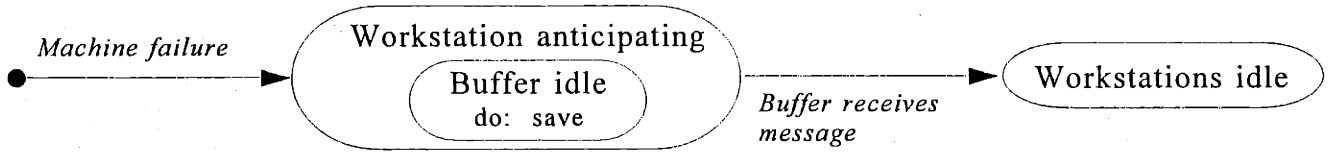
Architecture A, Action Scenario #12: Functional Model



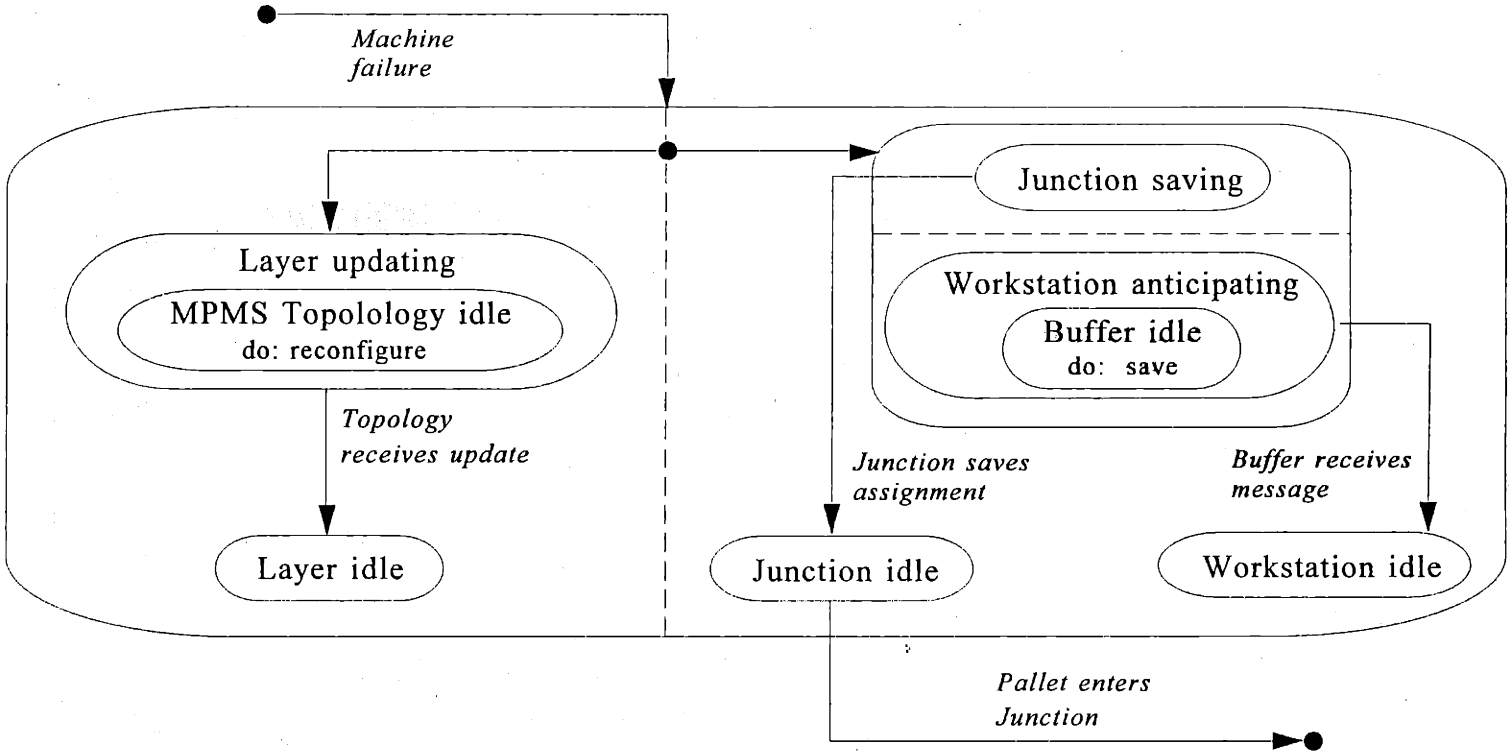
Architecture A, Action Scenario #13: Object Model



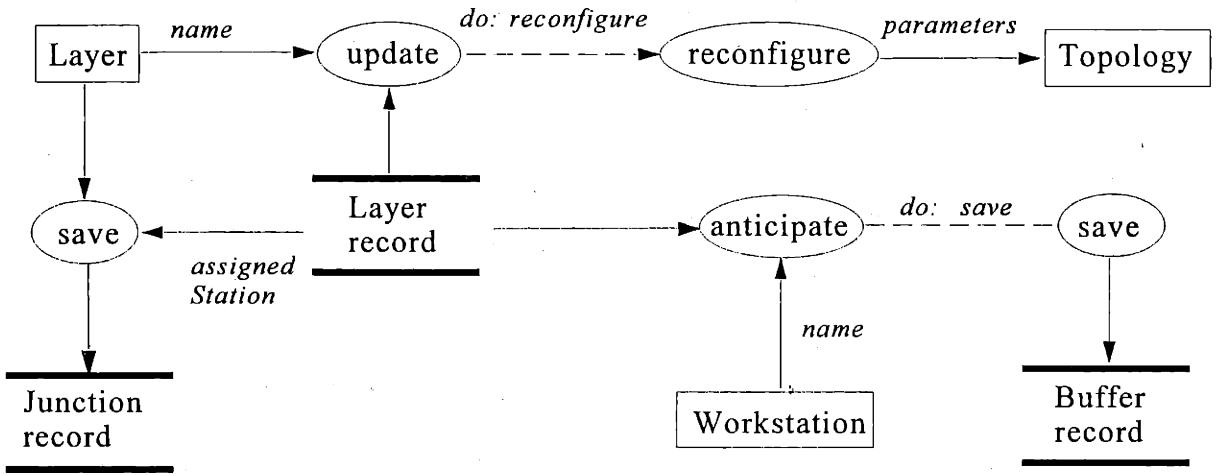




Architecture A, Action Scenario #13: Dynamic Model, Sheet 1



Architecture A, Action Scenario #13: Dynamic Model, Sheet 2



Architecture A, Action Scenario #13: Functional Model

Appendix B

Architecture B

In this appendix, we present a set of scenario script tables developed as an initial step towards obtaining the model for candidate Architecture B (pallet-driven/guideway-decision based). Tables B.1-13 each correspond (as described in Chapter 3) to a particular *action scenario*, or thought experiment regarding the system evolution under Architecture B. Table B.14 shows examples of the "workstation bidding procedure" which is a crucial part of Architecture B. Following the tables are shown the corresponding scenario-derived object analysis models that comprise the first-iteration model of Architecture B.

Initiator	Event	Participant	Method	Notes/Assumptions
Sample MPMS System	order setup	Order Or_1	create	system receives customer order; begin action scenario #1
Order Or_1	communicate order entry	Layers ^0L, ^1L, ^2L, ^3L	introduce	layers save required tasks pertaining to order's part type composition
Order Or_1	request output station assignment	Layer ^3L (output layer)	request	order needs output station assignment
Layer ^3L (output layer)	query output stations for order assignment	Output Stations O_1, O_2	query	---
Output Stations O_1, O_2	compute cost and return result	Layer ^3L (output layer)	cost	possible factors: output stream backlog on previous orders, order priority
Layer ^3L (output layer)	communicate choice for output station assignment	Output Station O_1, Order Or_1, Junctions $^3J_1$, $^3J_2$	choose	O_1 returns lowest result; output station saves order assignment; junctions save "output station list"
Layers ^1L, ^2L	initiate workstation bidding for outstanding tasks	Workstations $^1W_1$, $^1W_2$, $^2W_1$, $^2W_2$	initiate	includes *all* outstanding tasks of layer, not just tasks within particular order
Workstations $^1W_1$, $^1W_2$,, $^2W_1$, $^2W_2$	return "task preference list"	Layers ^1L, ^2L	bid*[1]	possible factors: machine setup, task processing time*[2]
Layers ^1L, ^2L	translate "task preference list" to "workstation preference list"	---	translate	---
Layers ^1L, ^2L	communicate "workstation preference list" resulting from bid	Junctions $^1J_1$, $^1J_2$, $^2J_1$, $^2J_2$	update	junctions save resulting "workstation preference list" for each task
Layer ^0L (input layer)	pallet setup	Input Stations I_1, I_2	initiate	end action scenario #1; begin action scenario #2

*[1] See Table B.14 for examples of bidding procedure.

*[2] If machine failure has occurred, workstation does not return "task preference list," end action scenario #1; begin action scenario #13.

Table B.1: Architecture B, Action Scenario #1

Initiator	Event	Participant	Method	Notes/Assumptions
Layer ^{0}L (input layer)	pallet setup	Input Stations I_1, I_2	initiate	end action scenario #1; begin action scenario #2
Input Stations I_1, I_2	return "task preference list"	Layer ^{0}L (input layer)	bid*[3]	possible factors: input stream setup, input task processing time
Layer ^{0}L (input layer)	translate "task preference list" to "input station preference list"	---	translate	input layer saves resulting "input station preference list" for each input task
Layer ^{0}L (input layer)	communicate choice for input station asignment	Input Station I_1	choose	part type A selected and "input station preference list" yields I_1
Input Station I_1	prepare raw pallet	Pallet P_{A1}	prepare	pallet tagged with part type A, task sequence TS_A, member Order Or_1
Pallet P_{A1}	raw pallet enters input station	Input Buffer Ib_1	introduce	end action scenario #2; begin action scenario #3

*[3] See Table B.14 for examples of bidding procedure.

Table B.2: Architecture B, Action Scenario #2

Initiator	Event	Participant	Method	Notes/Assumptions
Pallet P_{Al}	raw pallet enters input station	Input Buffer Ib_1	introduce	end action scenario #2; begin action scenario #3
Input Buffer Ib_1	communicate pallet waiting for input stream processing	Input Stream Is_1	update	input buffer empty prior to pallet arrival
Input Stream Is_1	communicate readiness for next pallet in input buffer queue	Input Buffer Ib_1	ready	input stream idle prior to pallet arrival
Pallet P_{Al}	raw pallet enters input stream	Input Stream Is_1	introduce	
Input Stream Is_1	perform input task on raw pallet	Pallet P_{Al}	perform	raw pallet loaded with raw material pertaining to particular part type
Input Stream Is_1	communicate successful task completion on pallet	Pallet P_{Al} Layer ^0L	complete	layer saves its record
Input Stream Is_1	communicate readiness for next raw pallet in input buffer queue	Input Buffer Ib_1	ready	---
Pallet P_{Al}	pallet enters link (with downstream junction)	Link $^1G_1$	introduce	end action scenario #3; begin action scenario #4 (Layer 1)

Table B.3: Architecture B, Action Scenario #3

Initiator	Event	Participant	Method	Notes/Assumptions
Pallet P_{A1}	pallet enters link (with downstream junction)	Link $^1G_1$	introduce	end action scenario #3; begin action scenario #4 (Layer 1)
Link $^1G_1$	communicate pallet entry	Junction $^1J_1$	update	link has downstream junction
Junction $^1J_1$	scan junction record for possible outgoing links	---	scan	junction record carries "workstation preference list" and route map
Junction $^1J_1$	query appropriate outgoing links for pallet assignment	Links $^1G_2$, $^1H_2$	query	need only query outgoing links leading to workstations that can do required task
Links $^1G_2$, $^1H_2$	compute cost and return result	Junction $^1J_1$	cost	possible factors: link traffic, link length to workstation, pallet priority
Junction $^1J_1$	make choice for outgoing link	---	choose	choice also depends on "workstation preference list"; $^1G_2$ returns lowest result
Pallet P_{A1}	pallet enters junction	Junction $^1J_1$	introduce	end action scenario #4; begin action scenario #5

Table B.4: Architecture B, Action Scenario #4

Initiator	Event	Participant	Method	Notes/Assumptions
Pallet P_{A1}	pallet enters junction	Junction $^1J_1$	introduce	end action scenario #4; begin action scenario #5
Junction $^1J_1$	route pallet	---	route	outgoing link assignment completed before pallet enters junction
Pallet P_{A1}	pallet enters link (with downstream workstation)	Link $^1G_2$	introduce	end action scenario #5; begin action scenario #6*[4]

*[4] If Workstation $^1W_2$ rather than $^1W_1$ had been chosen, then pallet would have been routed to Link $^1H_2$ which has a downstream junction, and action scenario #4 rather than #6 would begin.

Table B.5: Architecture B, Action Scenario #5

Initiator	Event	Participant	Method	Notes/Assumptions
Pallet P_{A1}	pallet enters link (with downstream workstation)	Link $^1G_2$	introduce	end action scenario #5; begin action scenario #6
Link $^1G_2$	communicate pallet entry	Workstation $^1W_1$	update	link has downstream workstation; buffer saves arrival
Pallet P_{A1}	pallet enters workstation	Buffer $^1B_1$	introduce	end action scenario #6; begin action scenario #7

Table B.6: Architecture B, Action Scenario #6

Initiator	Event	Participant	Method	Notes/Assumptions
Pallet P_{A1}	pallet enters workstation	Buffer $^1B_1$	introduce	end action scenario #6; begin action scenario #7
Buffer $^1B_1$	communicate pallet waiting for machine processing	Machine $^1M_1$	update	buffer empty prior to pallet arrival
Machine $^1M_1$	communicate readiness for next pallet in buffer queue	Buffer $^1B_1$	ready	machine idle prior to pallet arrival
Pallet P_{A1}	pallet enters machine	Machine $^1M_1$	introduce	buffer takes pallet off its record; machine sets up for required task
Machine $^1M_1$	perform required task on pallet	Pallet P_{A1}	perform	machine moves pallet into inspection station area upon completed task
Machine $^1M_1$	inspect part on pallet	Pallet P_{A1}	inspect*[5]	part on pallet passes inspection test
Machine $^1M_1$	communicate successful task completion on pallet	Pallet P_{A1} Layer ^1L	complete	layer saves its record
Machine $^1M_1$	communicate readiness for next pallet in buffer queue	Buffer $^1B_1$	ready	---
Pallet P_{A1}	pallet enters link (with downstream junction)	Link $^2G_1$	introduce	end action scenario #7 (Layer 1); begin action scenario #4 (Layer 2)

*[5] If a part failure has occurred, inspection fails, end action scenario #7; begin action scenario #12

Table B.7: Architecture B, Action Scenario #7

Initiator	Event	Participant	Method	Notes/Assumptions
Pallet P_{A1}	pallet enters link (with downstream junction)	Link $^3G_1$	introduce	end action scenario #7 (layer 2); begin action scenario #8
Link $^3G_1$	communicate pallet entry	Junction $^3J_1$	update	link has downstream junction
Junction $^3J_1$	scan junction record for possible outgoing links	---	scan	junction record carries "output station list" and route map
Junction $^3J_1$	record choice for outgoing link	---	choose	possible factors: link chosen that leads to output station assigned to pallet's order
Pallet P_{A1}	pallet enters junction	Junction $^3J_1$	introduce	end action scenario #8; begin action scenario #9

Table B.8: Architecture B, Action Scenario #8

Initiator	Event	Participant	Method	Notes/Assumptions
Pallet P_{A1}	pallet enters junction	Junction $^3J_1$	introduce	end action scenario #8; begin action scenario #9
Junction $^3J_1$	route pallet	---	route	outgoing link assignment completed before pallet enters junction
Pallet P_{A1}	pallet enters link (with downstream output station)	Link $^3G_2$	introduce	end action scenario #9; begin action scenario #10*[6]

*[6] If Output Station O_2 rather than O_1 had been chosen, then pallet would have been routed to Link $^3H_2$ which has a downstream junction, and action scenario #8 rather than #10 would begin.

Table B.9: Architecture B, Action Scenario #9

Initiator	Event	Participant	Method	Notes/Assumptions
Pallet P_{A1}	pallet enters link (with downstream output station)	Link $^3G_2$	introduce	end action scenario #9; begin action scenario #10
Link $^3G_2$	communicate pallet entry	Output Station O_1	update	link has downstream output station
Pallet P_{A1}	pallet enters output station	Output Buffer Ob_1	introduce	end action scenario #10; begin action scenario #11

Table B.10: Architecture B, Action Scenario #10

Initiator	Event	Participant	Method	Notes/Assumptions
Pallet P_{A1}	pallet enters output station	Output Buffer Ob_1	introduce	end action scenario #10; begin action scenario #11
Output Buffer Ob_1	communicate pallet waiting for output stream processing	Output Stream Os_1	update	output buffer empty prior to pallet arrival
Output Stream Os_1	communicate readiness for next pallet in output buffer	Output Buffer Ob_1	ready	output stream idle prior to pallet arrival
Pallet P_{A1}	pallet enters output stream	Output Stream Os_1	introduce	output buffer takes pallet off record; output stream sets up for output task
Output Stream Os_1	perform output task on pallet	Pallet P_{A1}	perform	unload finished part from pallet
Output Stream Os_1	communicate successful part type completion	Layer ^3L (output layer)	complete	layer saves its record
Output Stream Os_1	communicate readiness for next pallet in output buffer queue	Output Buffer Ob_1	ready	---
Layer ^3L (output layer)	communicate part completion	Order Or_1	part_complete	order saves its record end action scenario #11*[7]
*[7] if completed part is final part in order, scenario continues...				
Layer ^3L (output layer)	communicate order completion	Output Station O_1, Order Or_1	order_complete	end action scenario #11; output staion removes order from record

Table B.11: Architecture B, Action Scenario #11

Initiator	Event	Participant	Method	Notes/Assumptions
Machine ${}^{1}M_{1}$	inspect part on pallet	Pallet P_{A1}	inspect*[8]	part on pallet fails inspection test
Machine ${}^{1}M_{1}$	communicate unsuccessful process completion on pallet	Layer ${}^{1}L$, Pallet P_{A1}	incomplete	pallet requires task of "fix fault"
layers and order coordinate for part failure recovery				
Layer ${}^{1}L$	communicate need for faulty pallet replacement	Order Or_1	replace	---
Order Or_1	communicate replacement order entry	Layers ${}^{0}L$, ${}^{1}L$, ${}^{2}L$, ${}^{3}L$	update	replacement order placed with higher priority;
Layer ${}^{0}L$ (input layer)	pallet setup	Input Station I_1, I_2	query	pallet requested with higher priority; begin action scenario #2 (with new pallet)
faulty pallet is redirected to a "fault bin"				
Machine ${}^{1}M_{1}$	communicate readiness for next pallet in buffer queue	Buffer ${}^{1}B_{1}$	ready	---
Pallet P_{A1}	pallet enters link (with downstream junction)	Link ${}^{2}G_{1}$	introduce (faulty)	pallet requires task of "fix fault"; begin action scenario #4 (Layer 2)
end action scenario #12				

*[8] Action scenario #7 ends early due to part failure.(inspection test fails), end action scenario #7; begin action scenario #12.

Table B.12: Architecture B, Action Scenario #12

Initiator	Event	Participant	Method	Notes/Assumptions
Workstations ${}^1W_1, {}^1W_2, {}^2W_1, {}^2W_2$	return "task preference list"	Layers ${}^1L$, ${}^2L$	bid*[9]	possible factors: machine setup, task processing time*[10]
Layers ${}^1L$, ${}^2L$	translate "task preference list" to "workstation preference list"	---	translate	---
Layer ${}^1L$	layer communicates machine failure to system	Sample MPMS Topology	notify	system reconfigures topology to account for effect of machine failure
Layers ${}^1L$, ${}^2L$	communicate "workstation preference list" resulting from	Junctions ${}^1J_1, {}^1J_2, {}^2J_1, {}^2J_2$	update	junctions save "workstation preference list" for each task
Layer ${}^0L$ (input layer)	pallet setup	Input Stations I_1, I_2	initiate	end action scenario #1; begin action scenario #2

*[9] See Table B.14 for examples of bidding procedure.

*[10] Action scenario #1 ends early due to machine failure (${}^1W_1$ does not return "workstation preference list"); begin action scenario #13

Table B.13: Architecture B, Action Scenario #13

Action Scenario #1

Layer ^{1}L: Task T_1 ⇒ initiate ⇒

Workstation	Task Preference List
$^{1}W_1$	T_1
$^{1}W_2$	T_1

⇒ translate ⇒

Task	Workstation Preference List
T_1	$^{1}W_1$ or $^{1}W_2$
T_1	$^{1}W_1$ or $^{1}W_2$

Layer ^{2}L: Task T_2, T_3 ⇒ initiate ⇒

Workstation	Task Preference List
$^{2}W_1$	T_3, T_4
$^{2}W_2$	T_4, T_3

⇒ translate ⇒

Task	Workstation Preference List
T_3	$^{2}W_1$, $^{2}W_2$
T_4	$^{2}W_2$, $^{2}W_1$

Action Scenario #2

Input Layer ^{0}L: Task $^{0}T_A$, $^{0}T_B$ ⇒ initiate ⇒

Input Station	Task Preference List
I_1	$^{0}T_A$, $^{0}T_B$
I_2	$^{0}T_A$, $^{0}T_B$

⇒ translate ⇒

Task	Input Station Preference List
$^{0}T_A$	I_1 or I_2
$^{0}T_B$	I_1 or I_2

Action Scenario #13

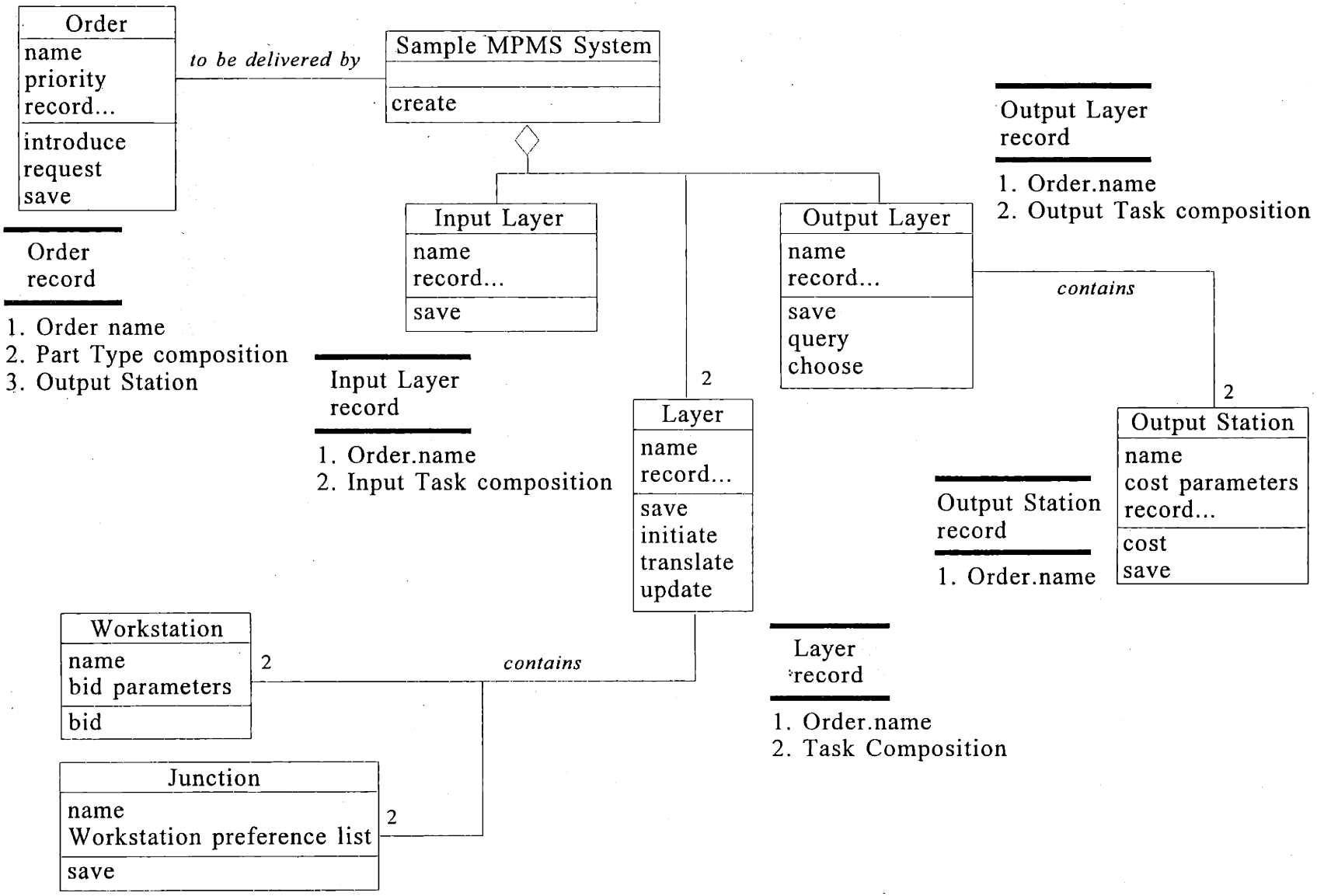
Layer ^{1}L: Task T_1 ⇒ initiate ⇒

Workstation	Task Preference List
$^{1}W_1$	---
$^{1}W_2$	T_1

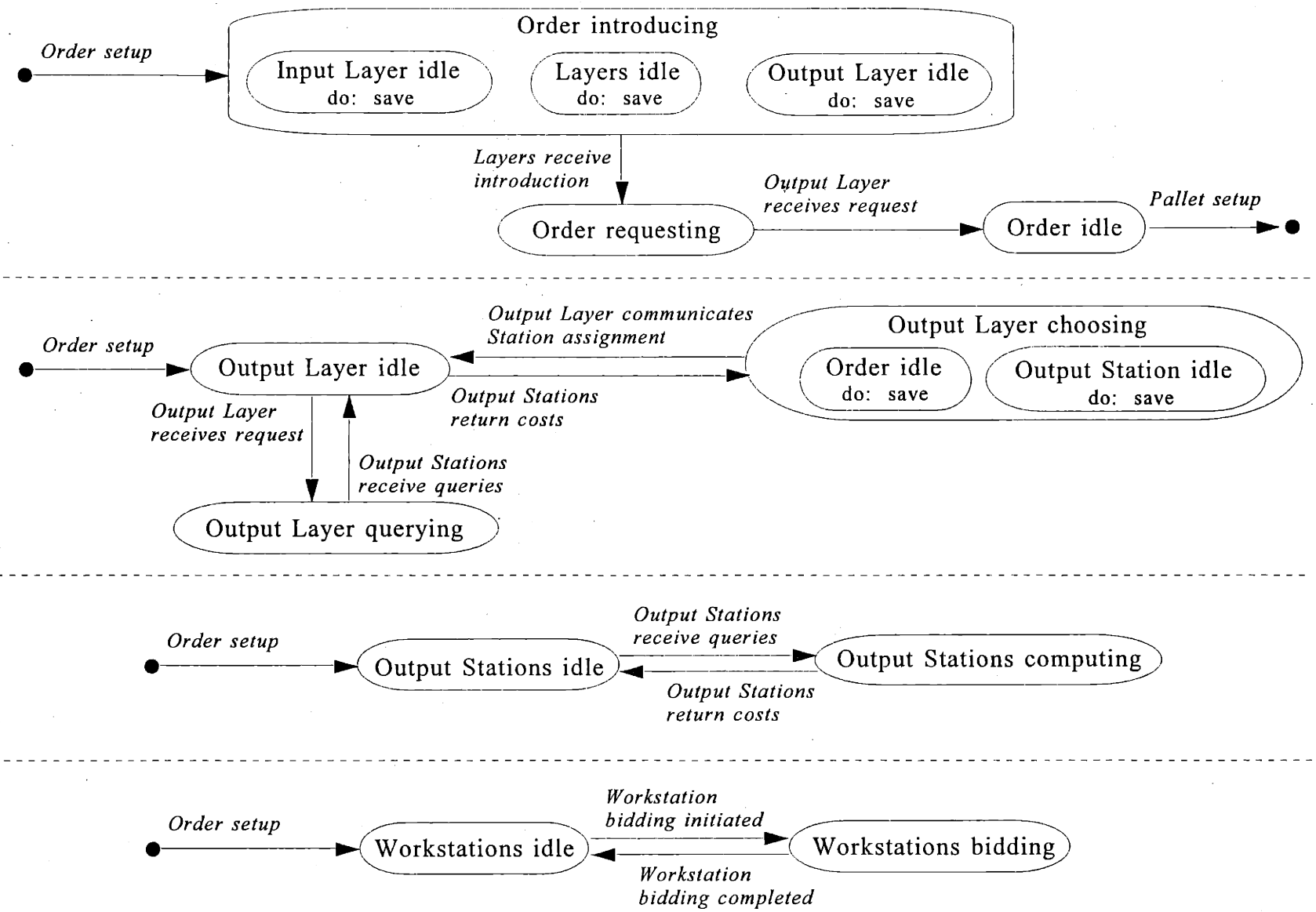
⇒ translate ⇒

Task	Workstation Preference List
T_1	$^{1}W_2$
T_1	$^{1}W_2$

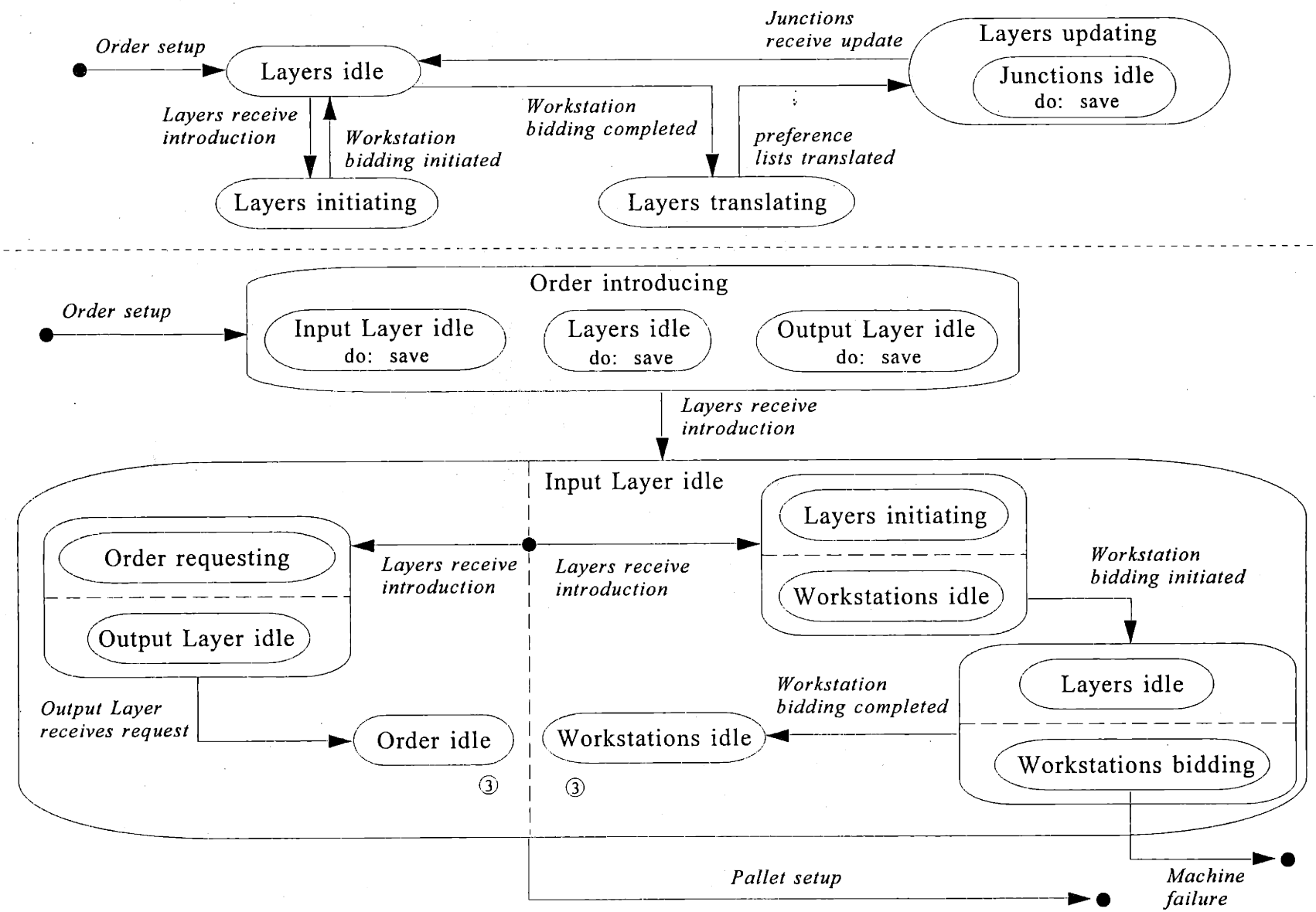
Table B.14: Architecture B, Examples of Bidding Procedure for Action Scenario #1, #2, and #13



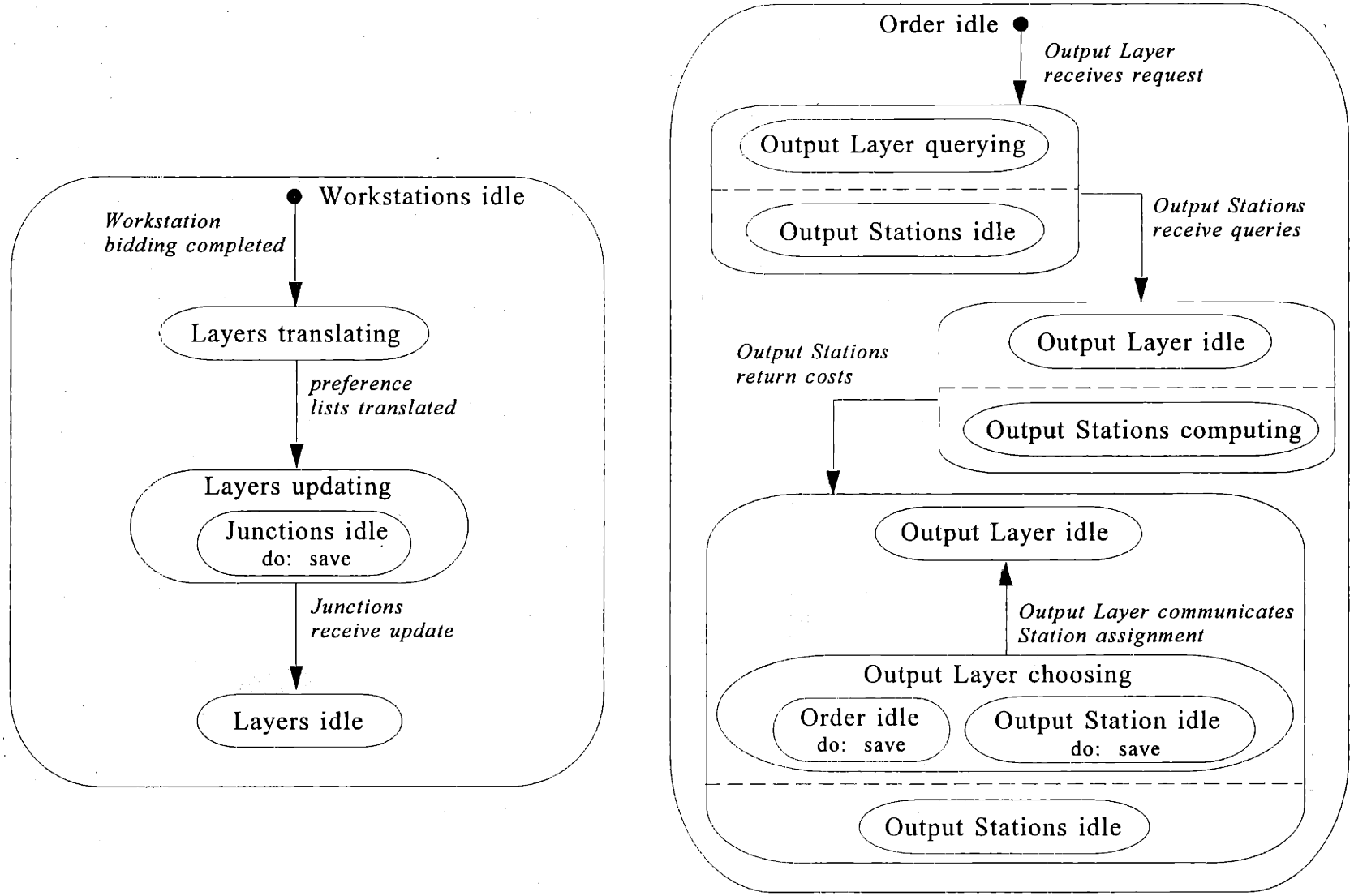
Architecture B, Action Scenario #1: Object Model



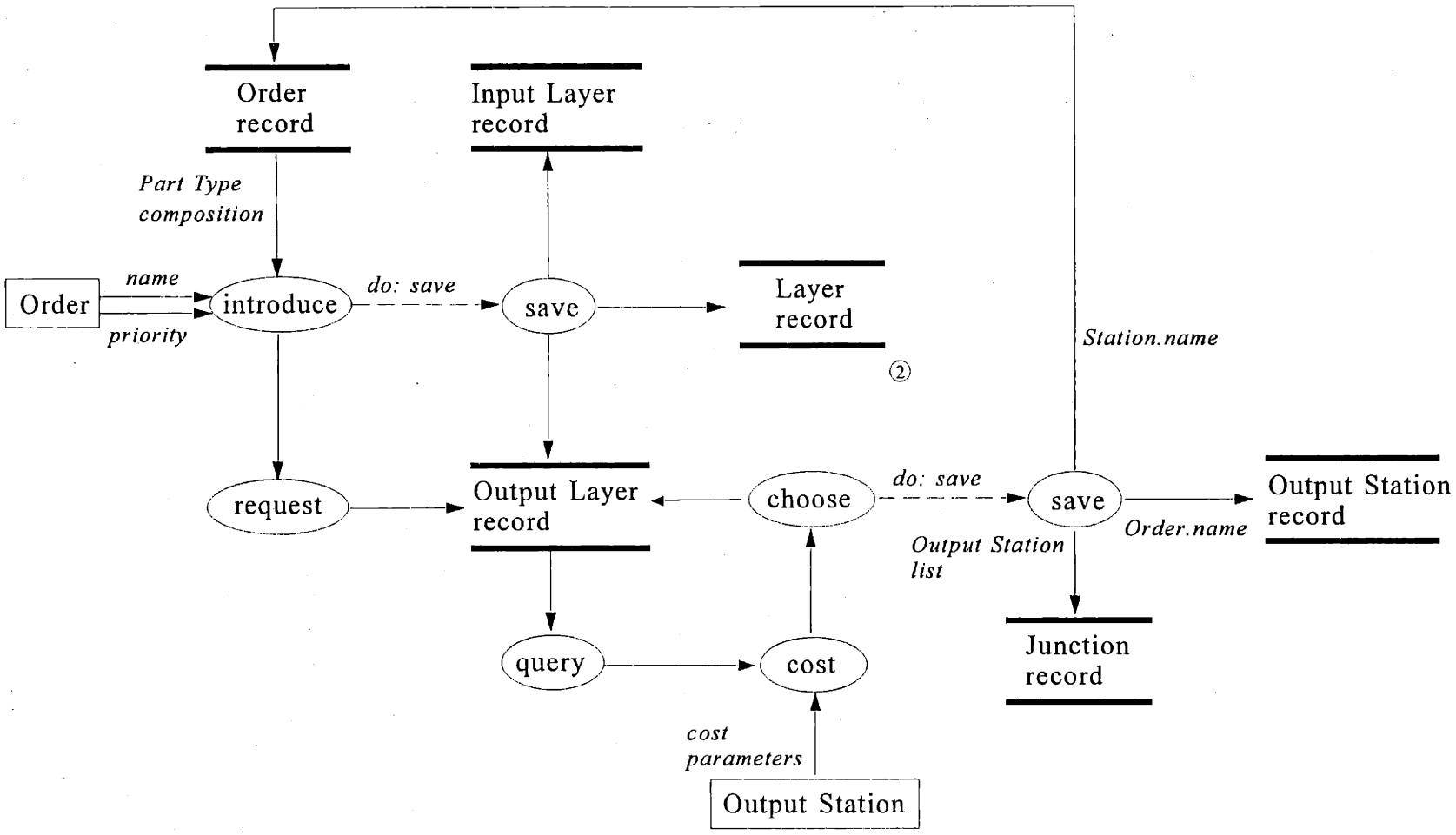
Architecture B, Action Scenario #1: Dynamic Model, Sheet 1



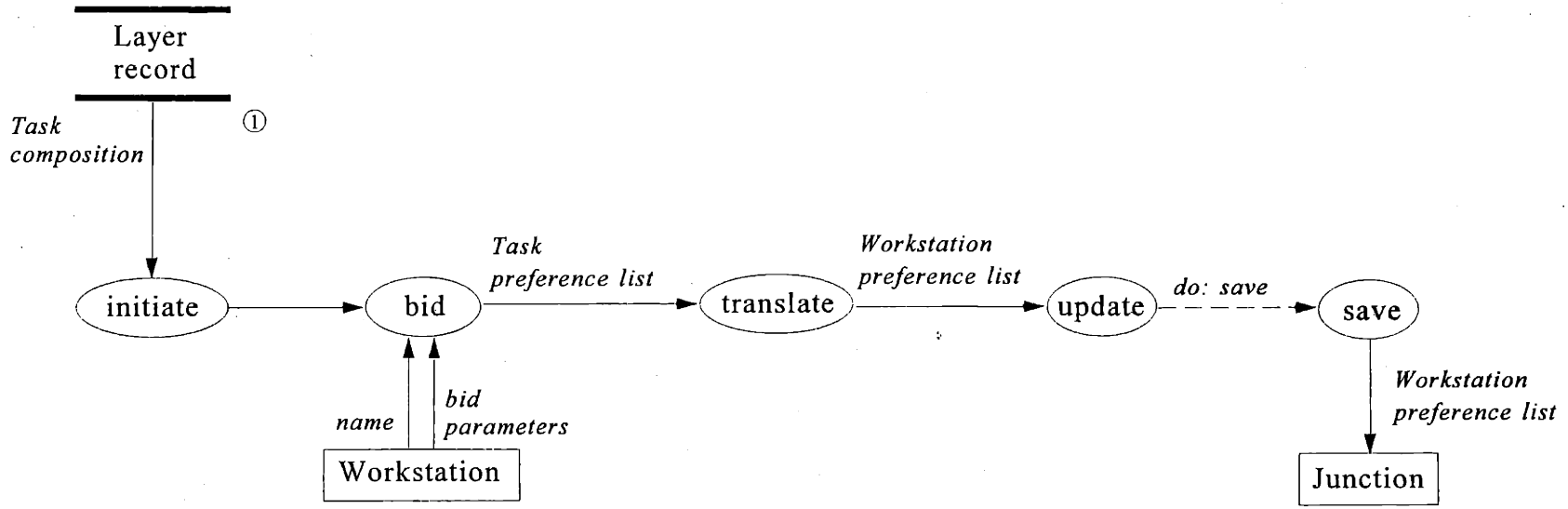
Architecture B, Action Scenario #1: Dynamic Model, Sheet 2



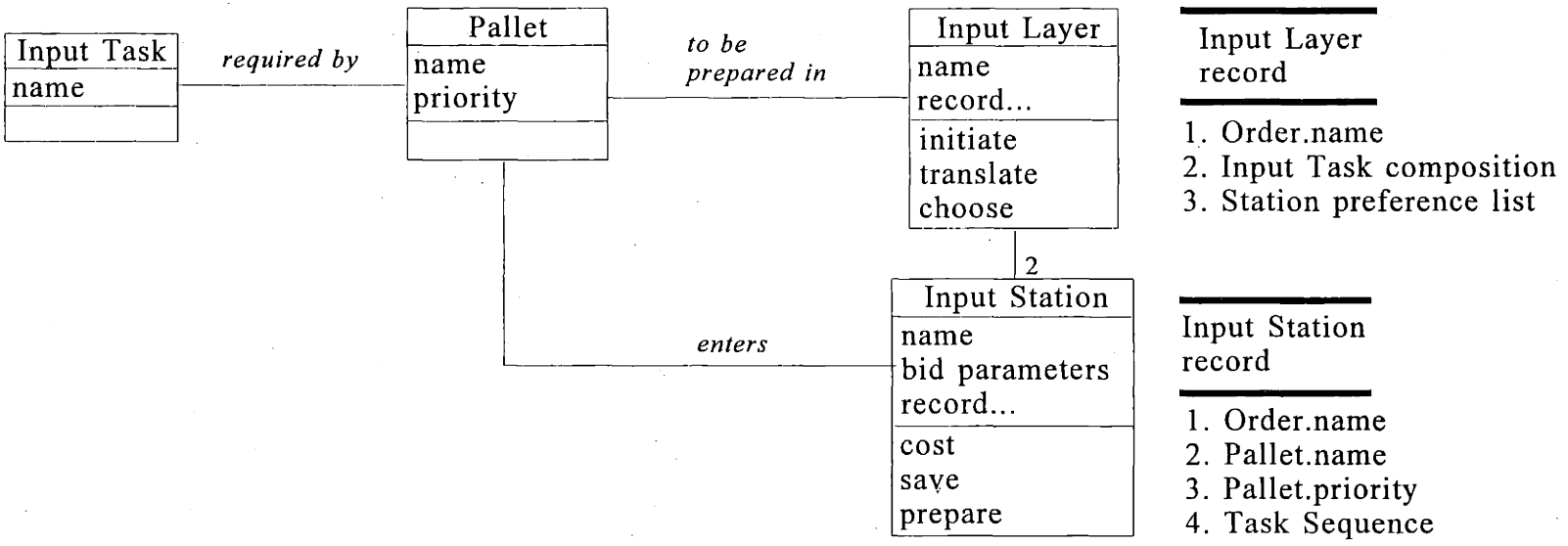
Architecture B, Action Scenario #1: Dynamic Model, Sheet 3



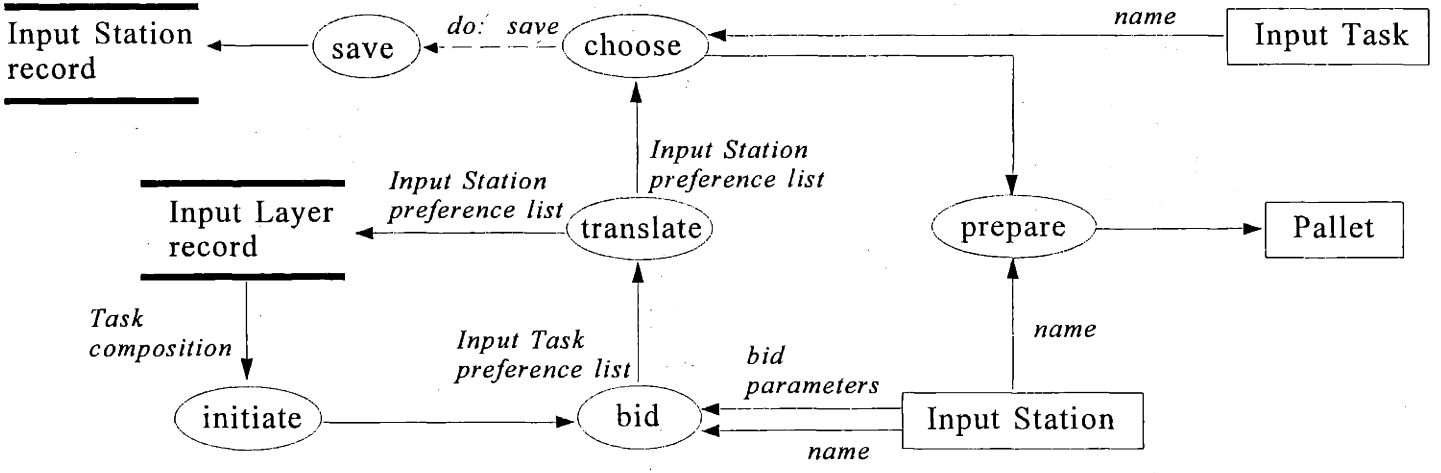
Architecture B, Action Scenario #1: Functional Model, Sheet 1



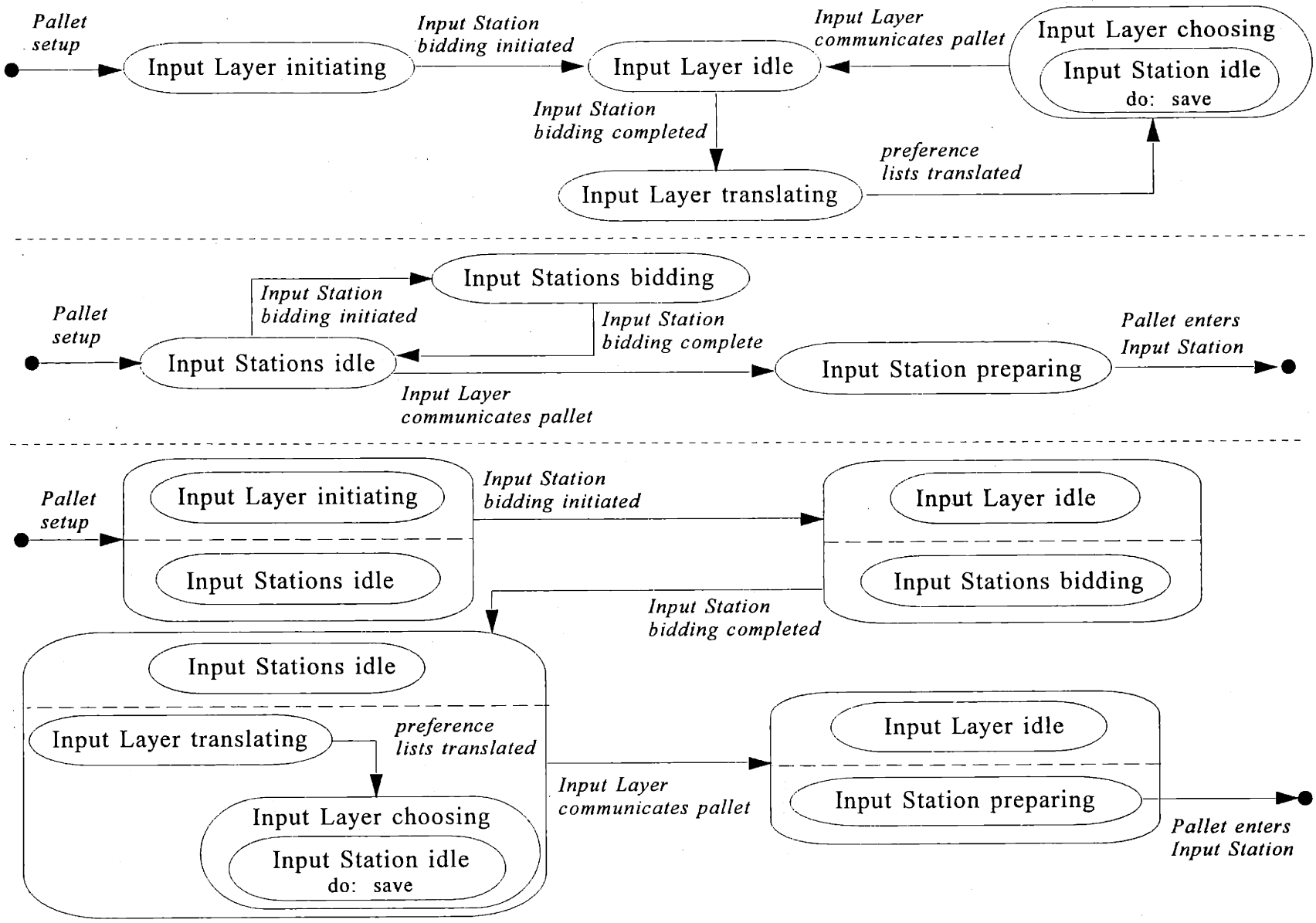
Architecture B, Action Scenario #1: Functional Model, Sheet 2



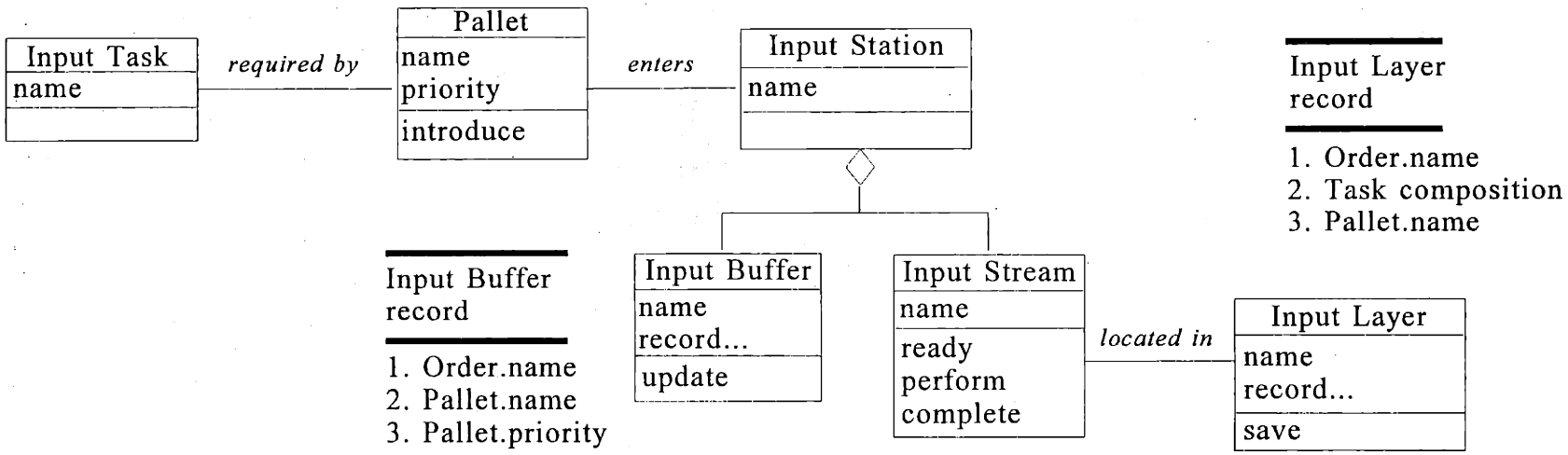
Architecture B, Action Scenario #2: Object Model



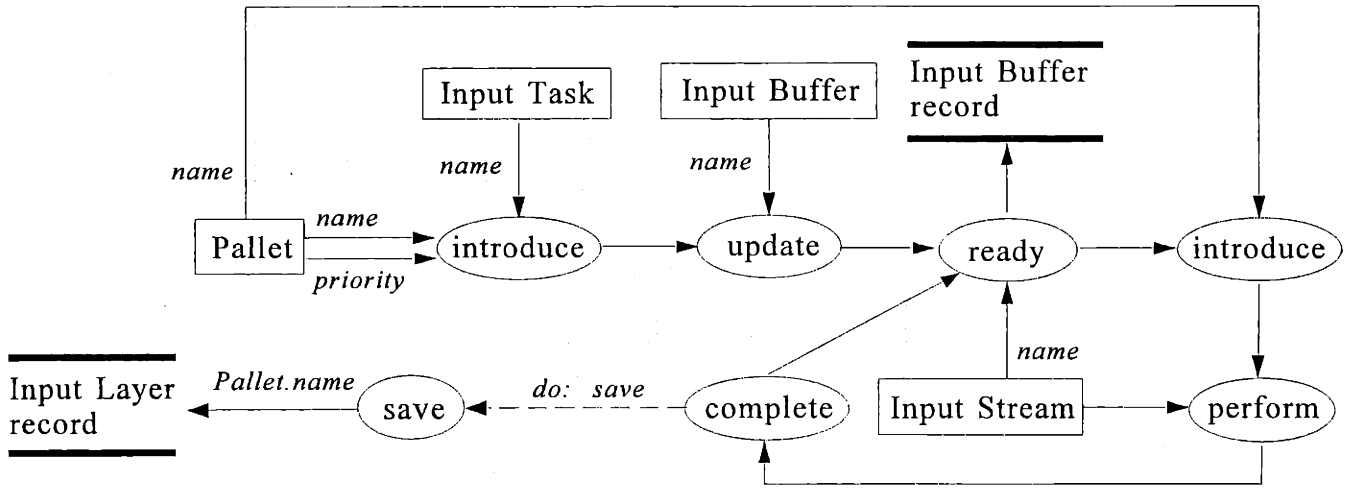
Architecture B, Action Scenario #2: Functional Model



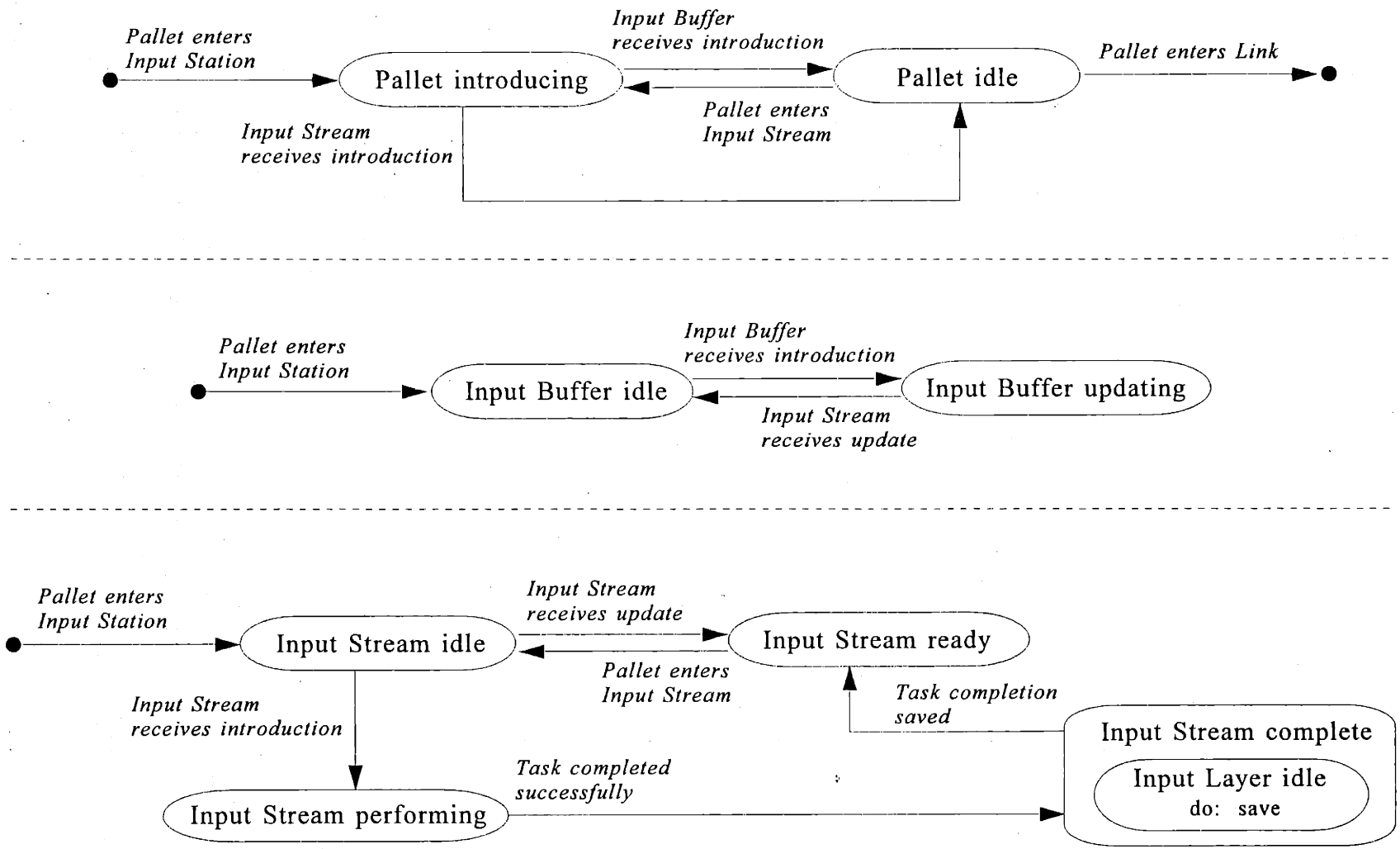
Architecture B, Action Scenario #2: Dynamic Model



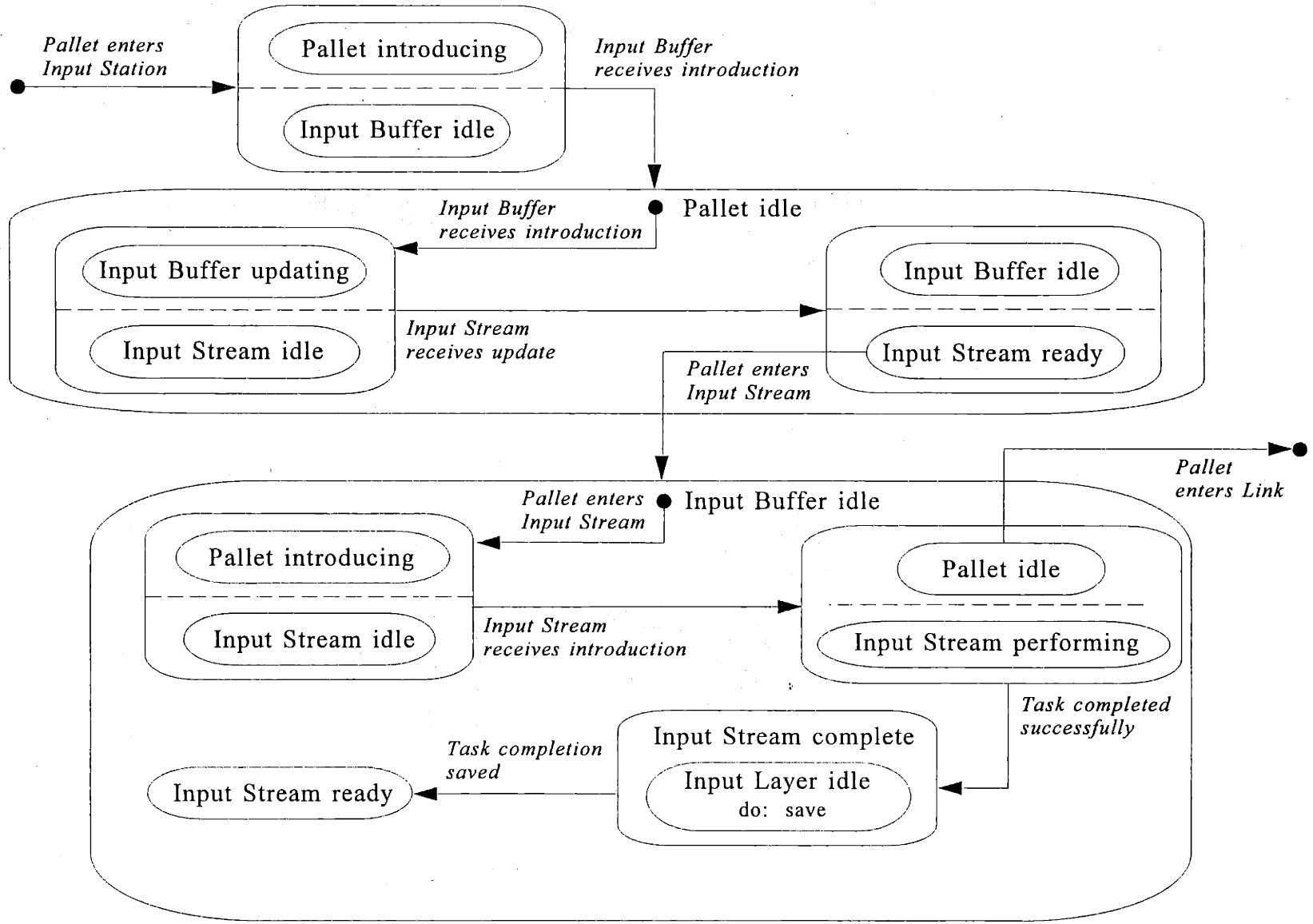
Architecture B, Action Scenario #3: Object Model



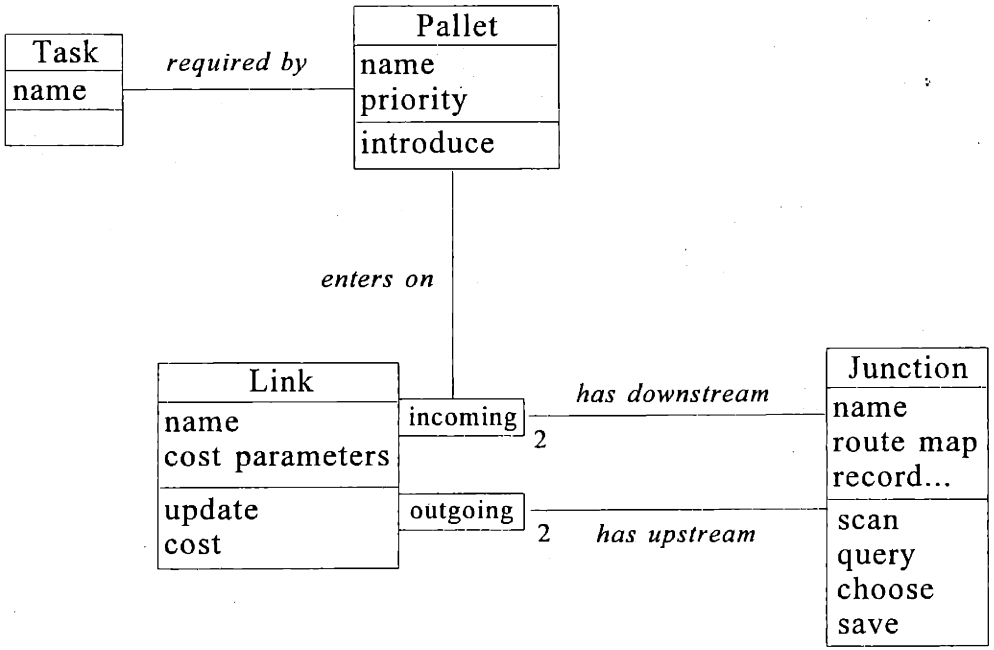
Architecture B, Action Scenario #3: Functional Model



Architecture B, Action Scenario #3: Dynamic Model, Sheet 1



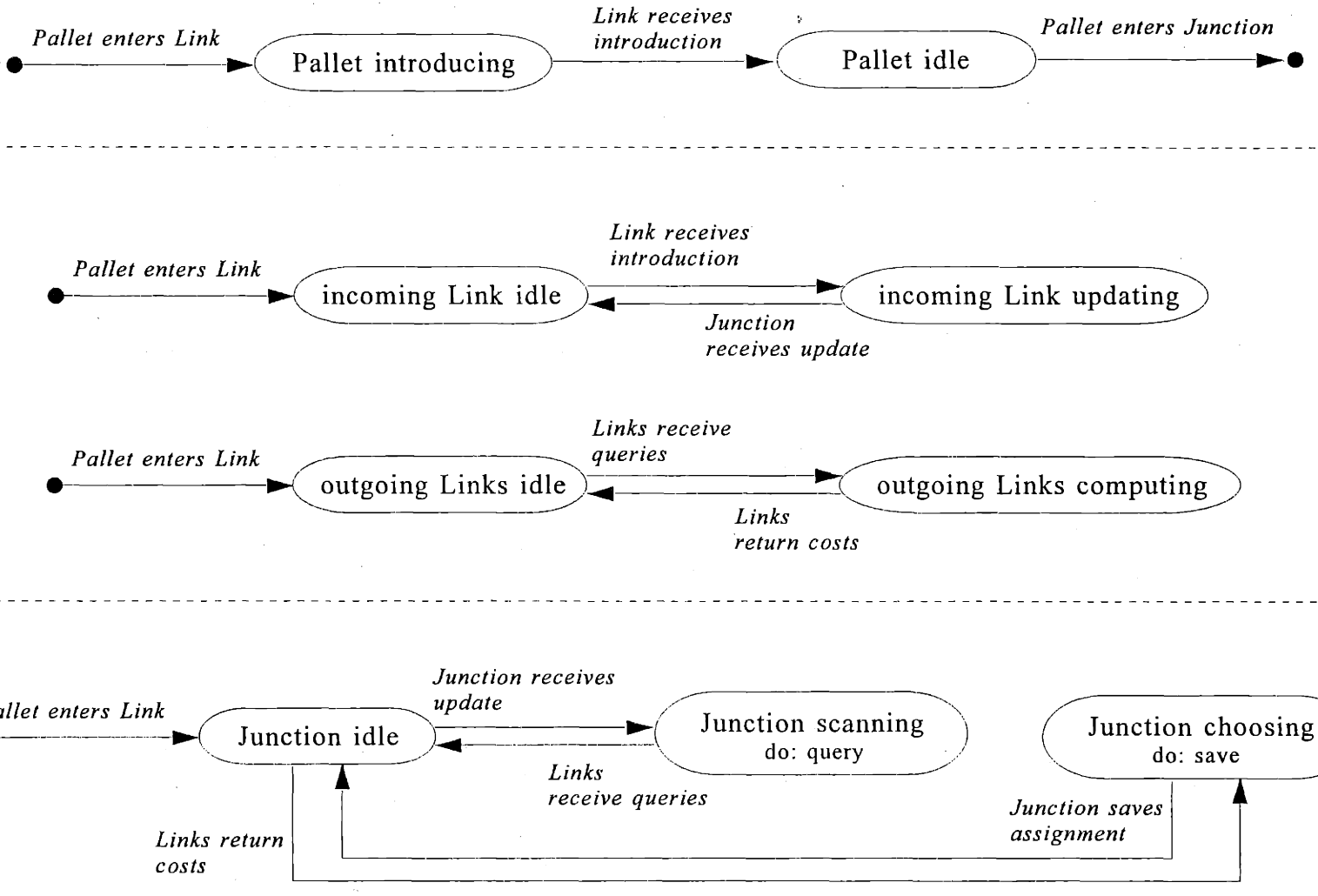
Architecture B, Action Scenario #3: Dynamic Model, Sheet 2



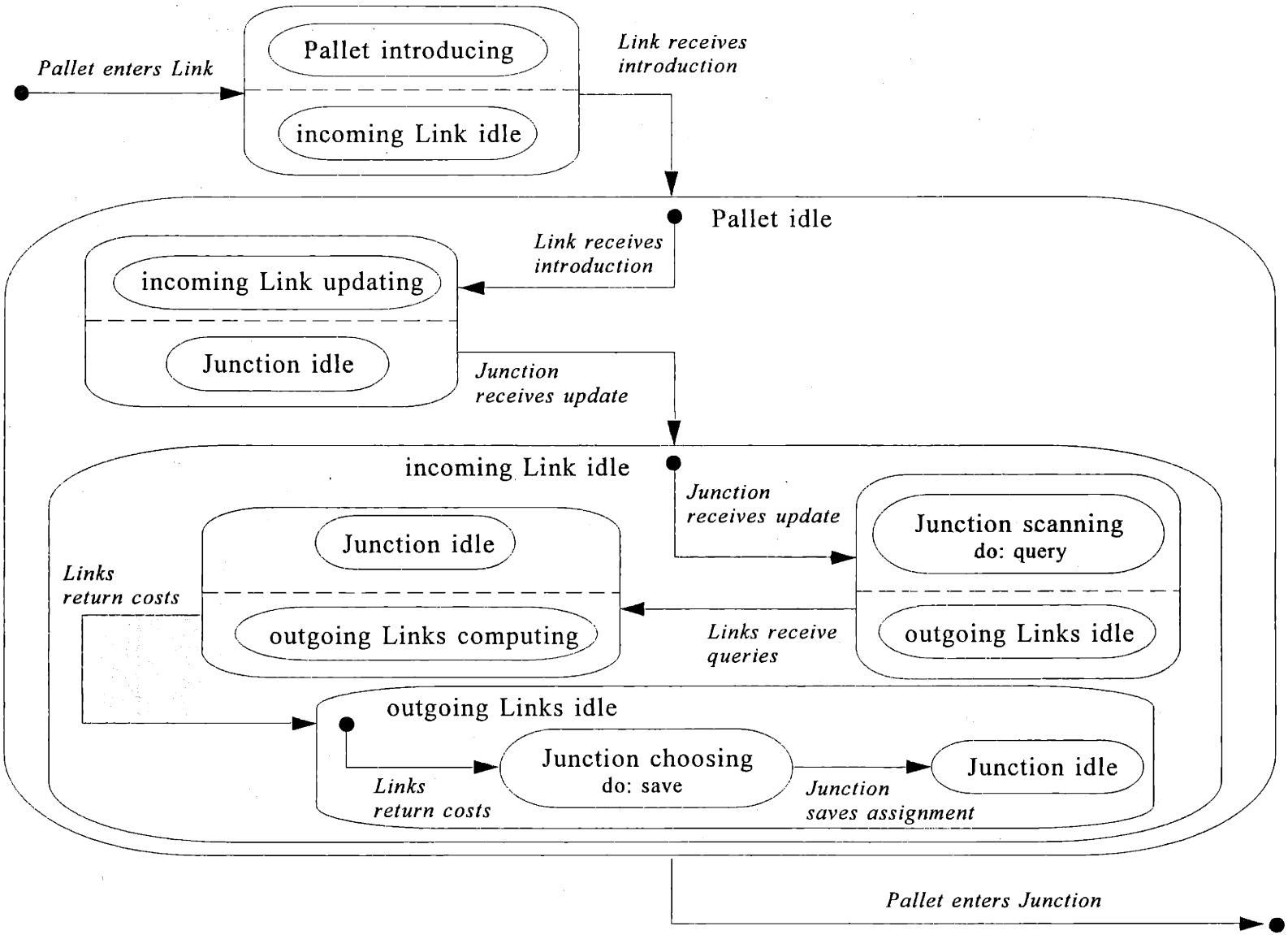
Junction
record

1. incoming Link.name
2. Pallet.name
3. Pallet.priority
4. Task.name
5. "Station Preference List"
6. outgoing Link.name

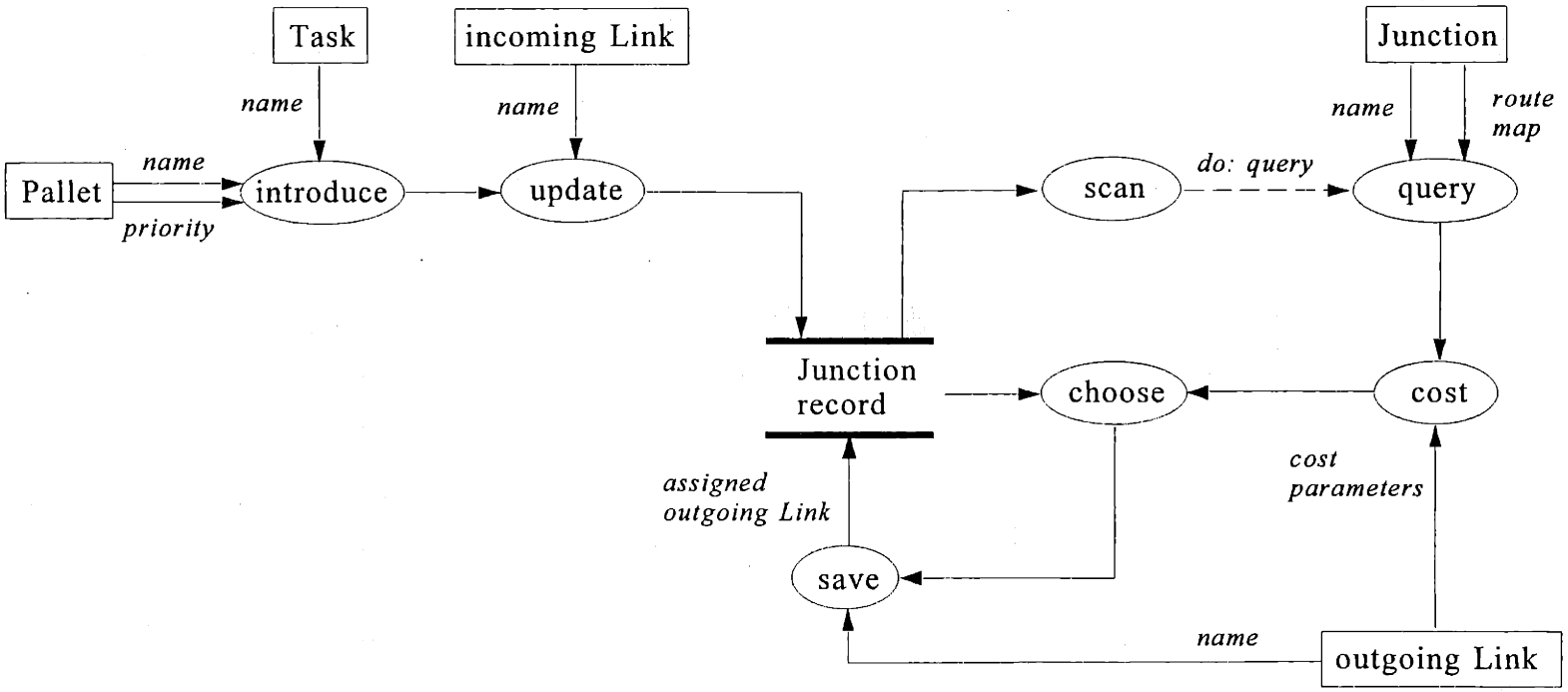
Architecture B, Action Scenario #4: Object Model



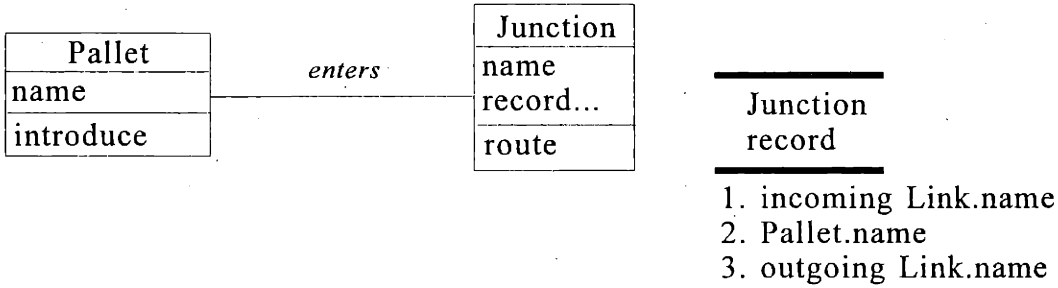
Architecture B, Action Scenario #4: Dynamic Model, Sheet 1



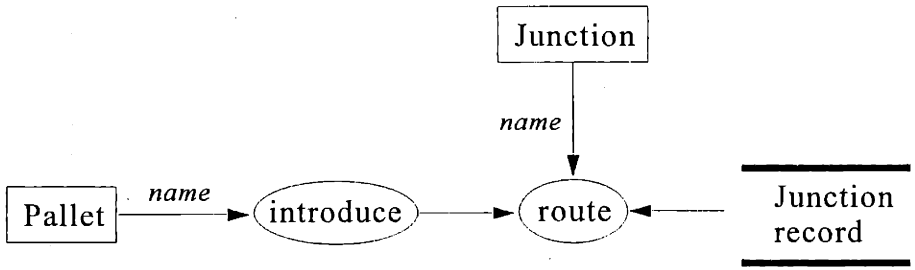
Architecture B, Action Scenario #4: Dynamic Model, Sheet 2



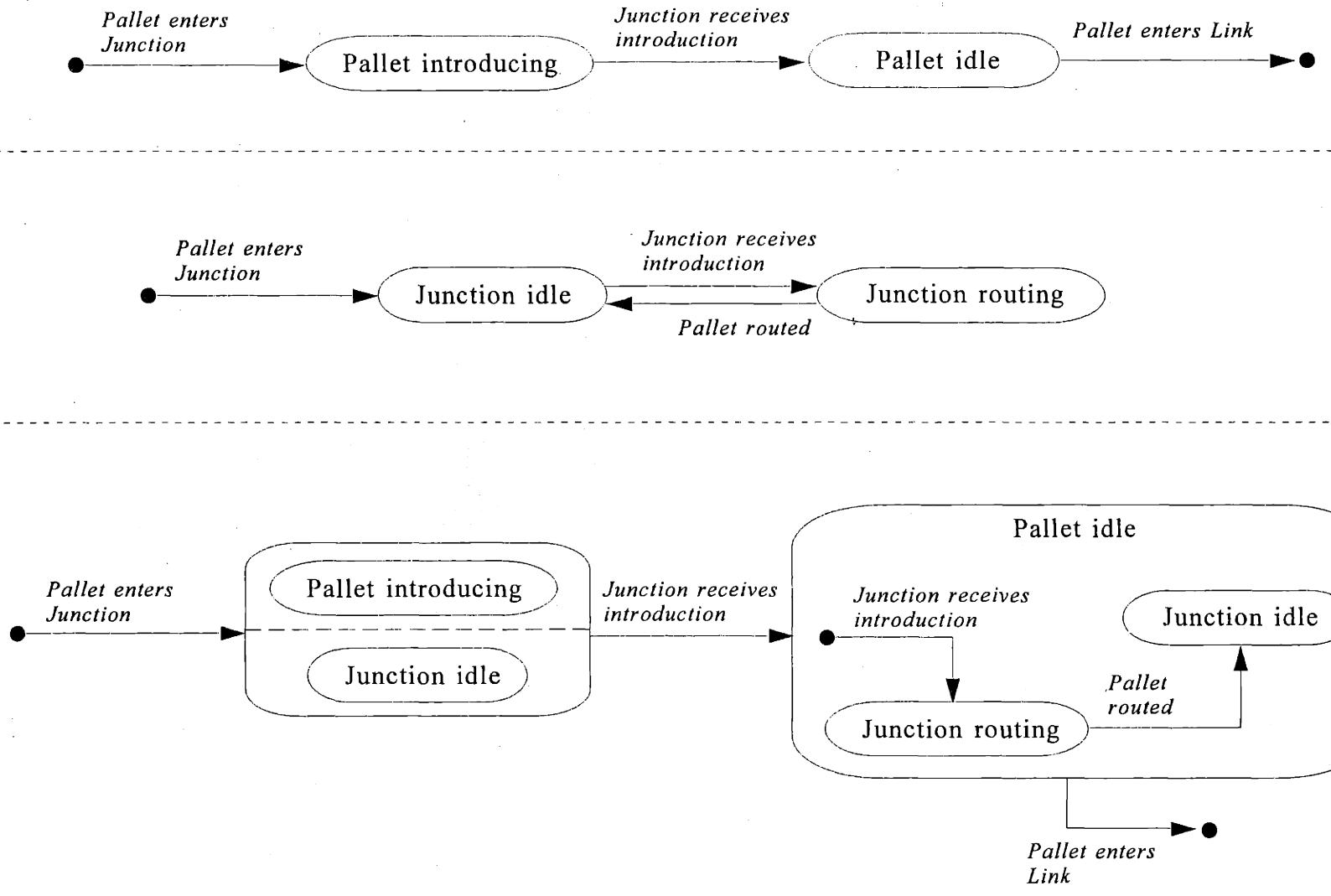
Architecture B, Action Scenario #4: Functional Model



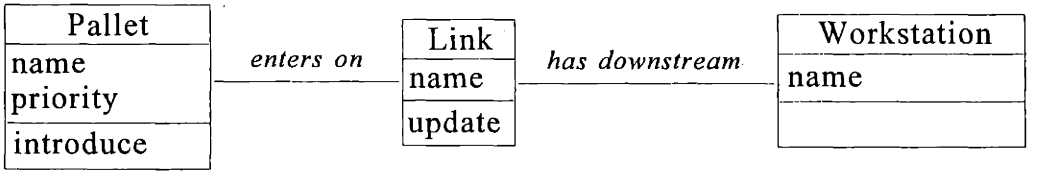
Architecture B, Action Scenario #5: Object Model



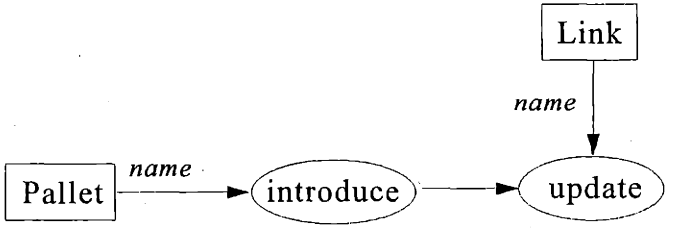
Architecture B, Action Scenario #5: Functional Model



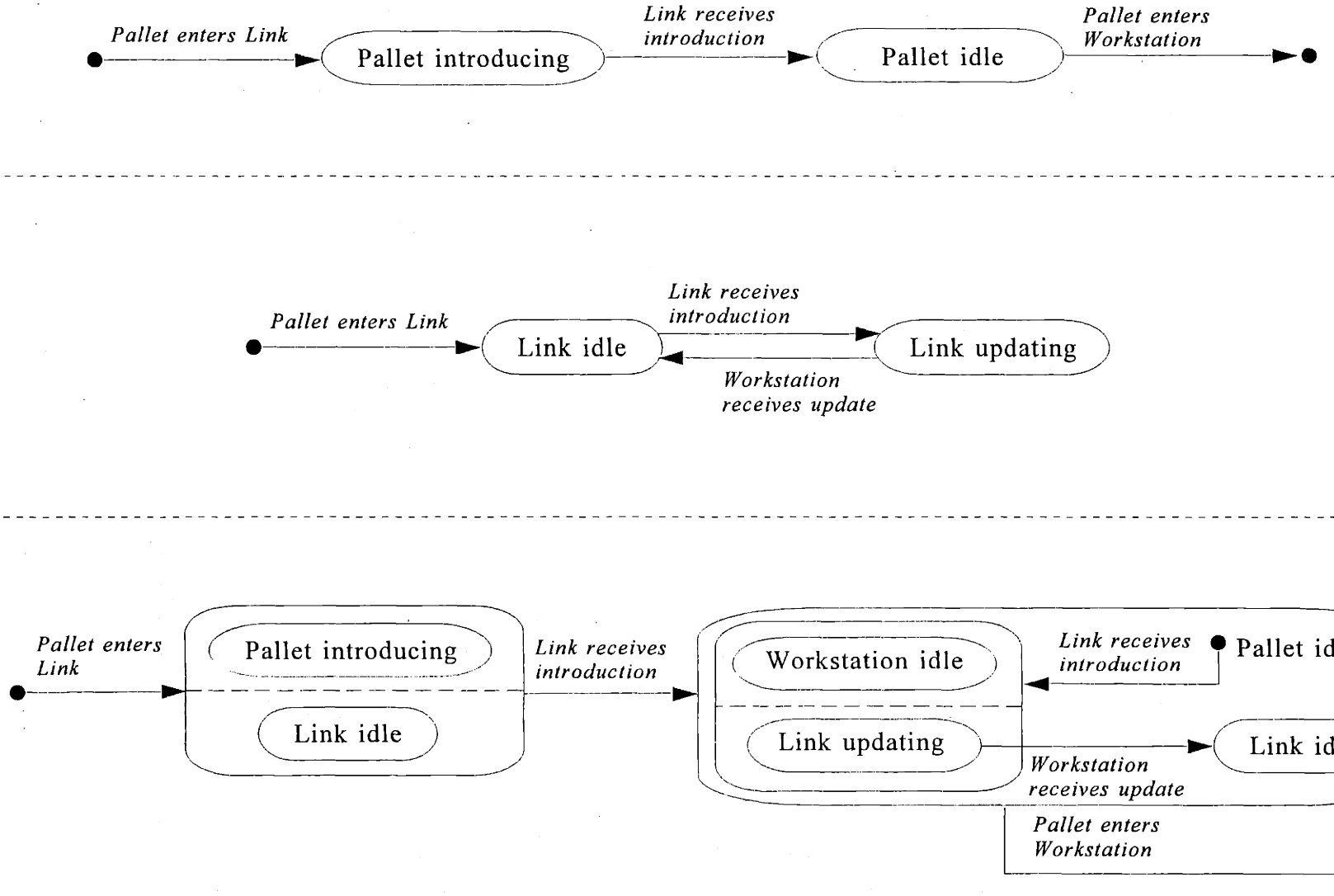
Architecture B, Action Scenario #5: Dynamic Model



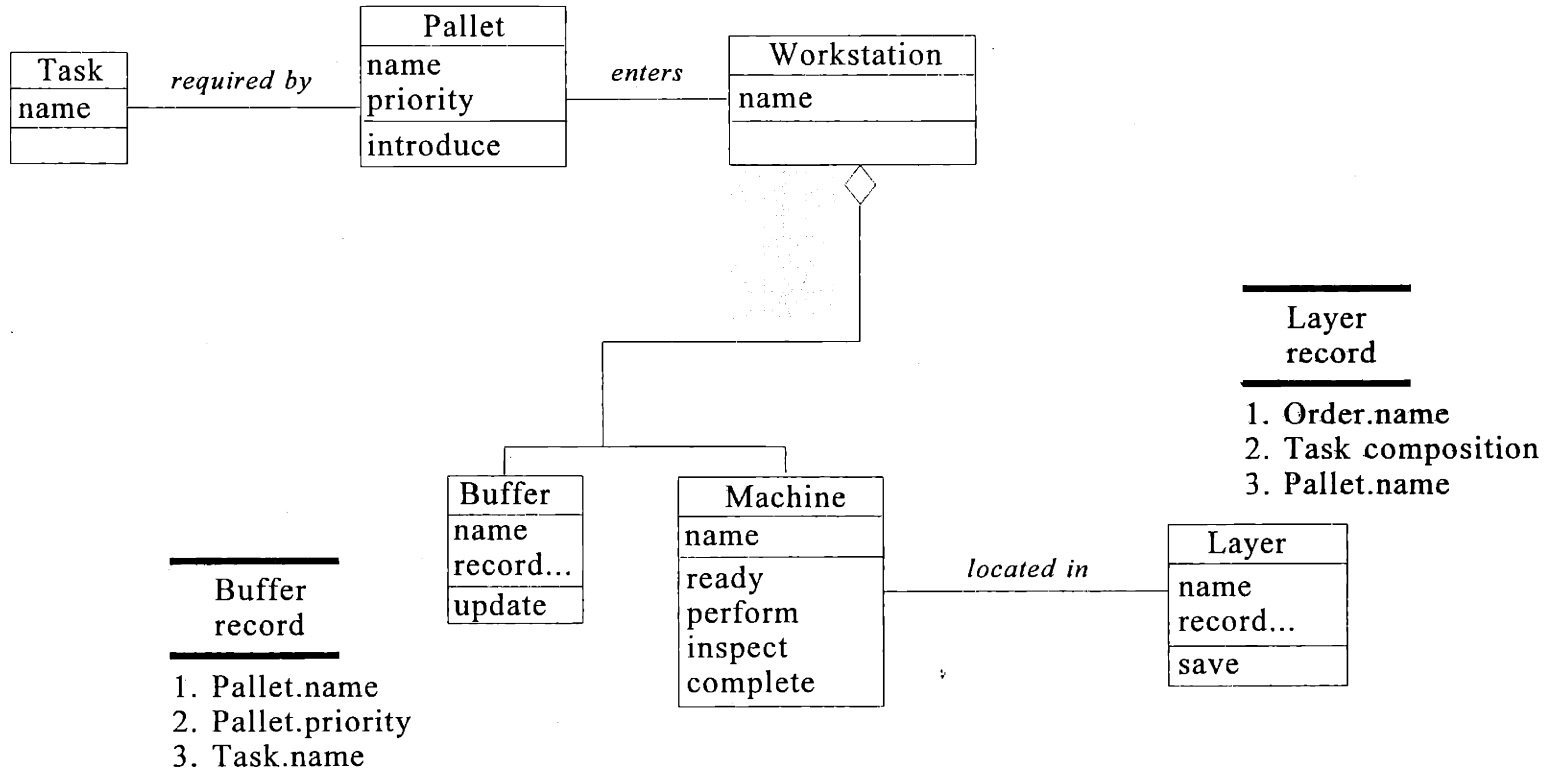
Architecture B, Action Scenario #6: Object Model



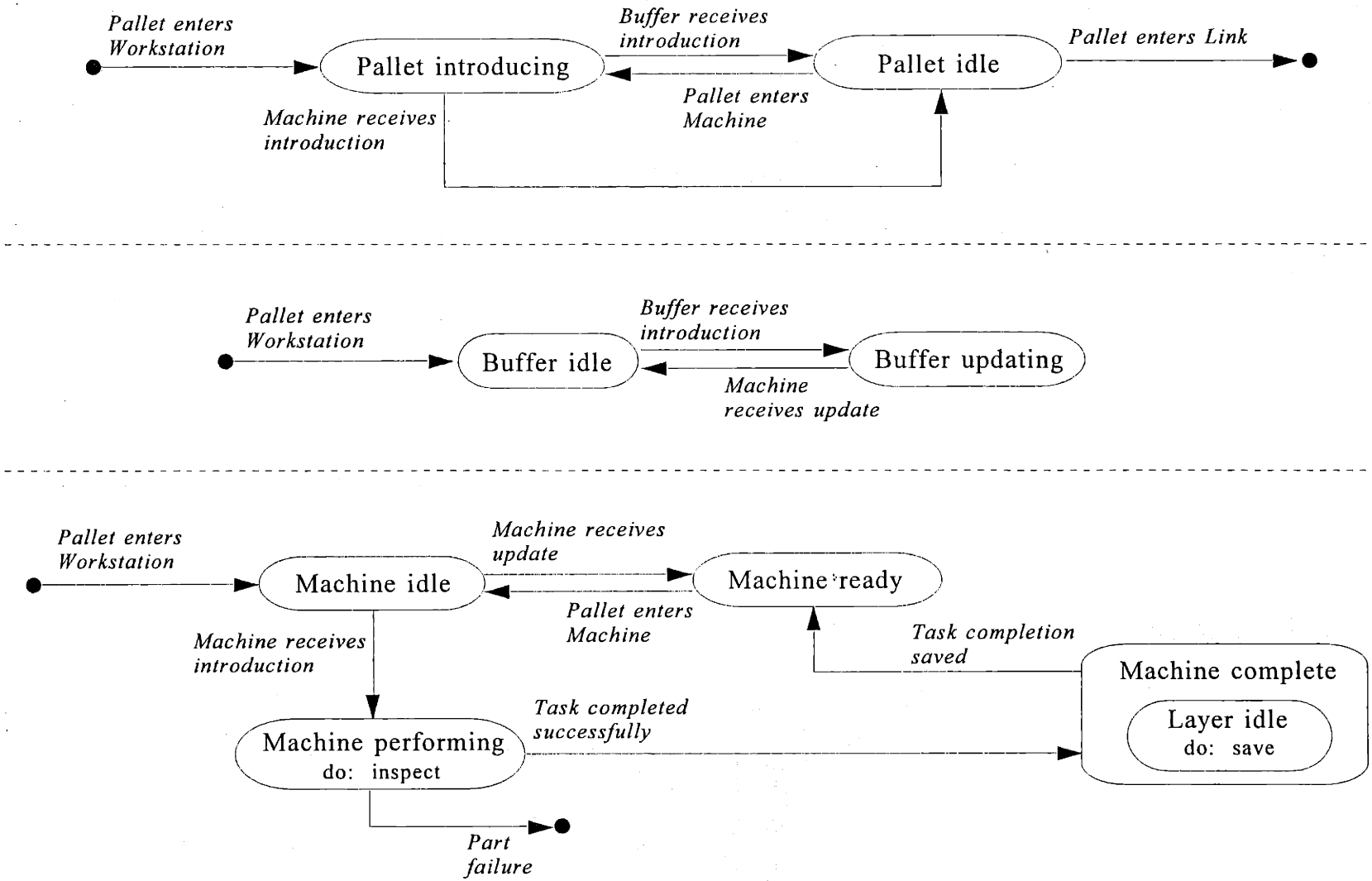
Architecture B, Action Scenario #6: Functional Model



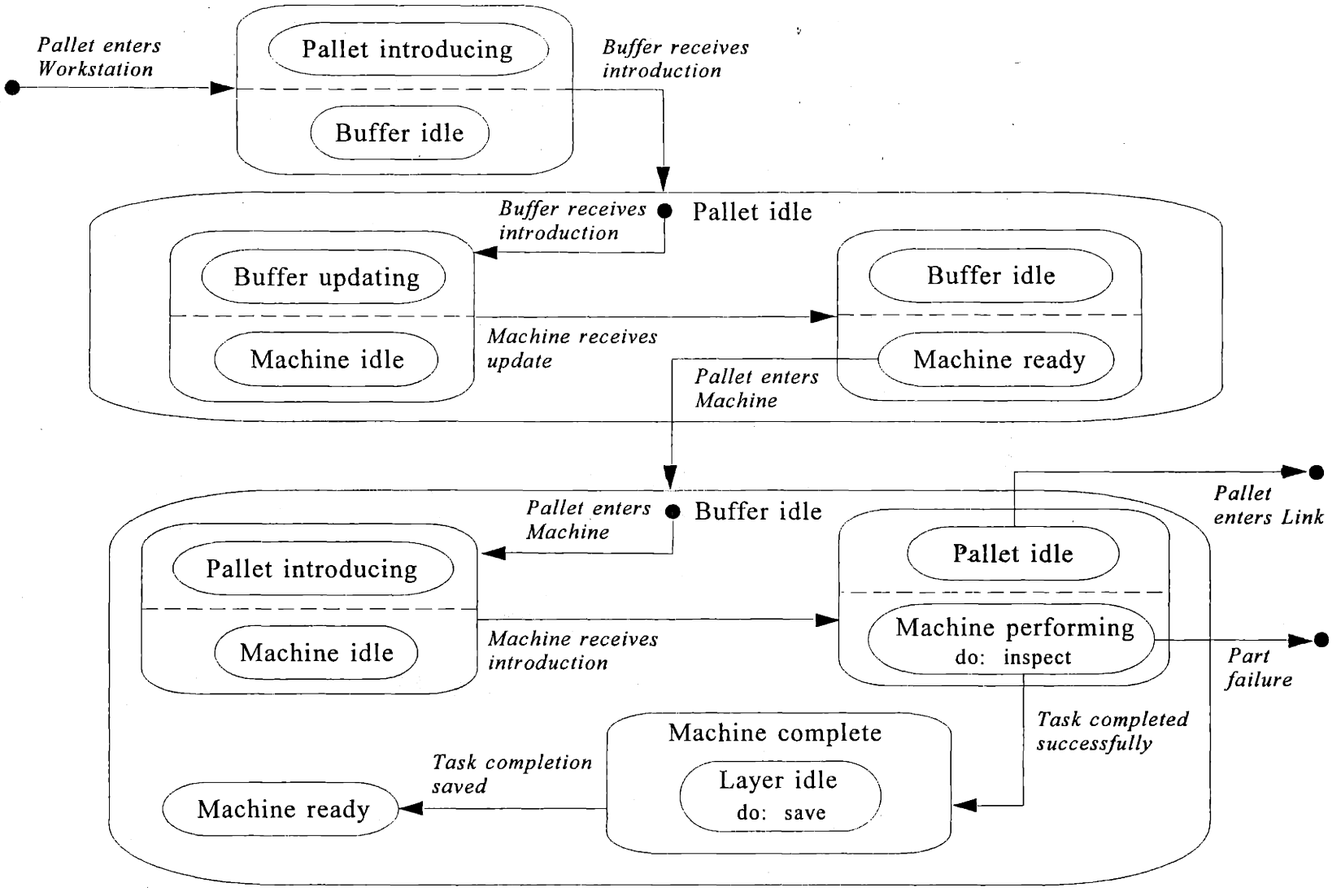
Architecture B, Action Scenario #6: Dynamic Model



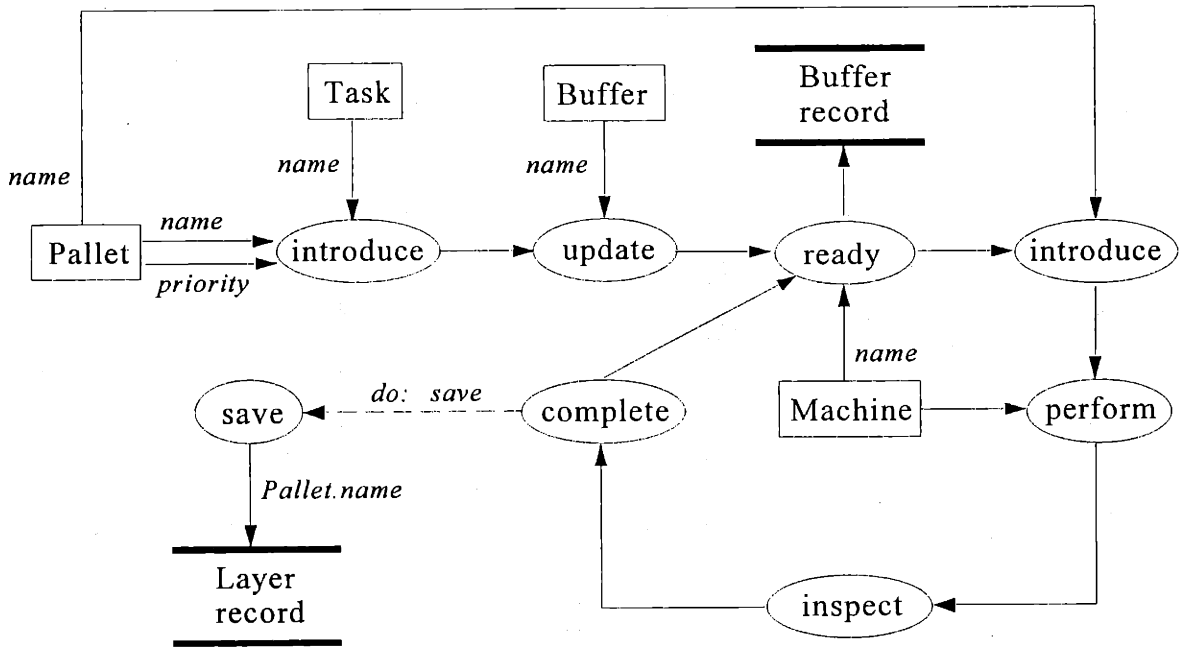
Architecture B, Action Scenario #7: Object Model



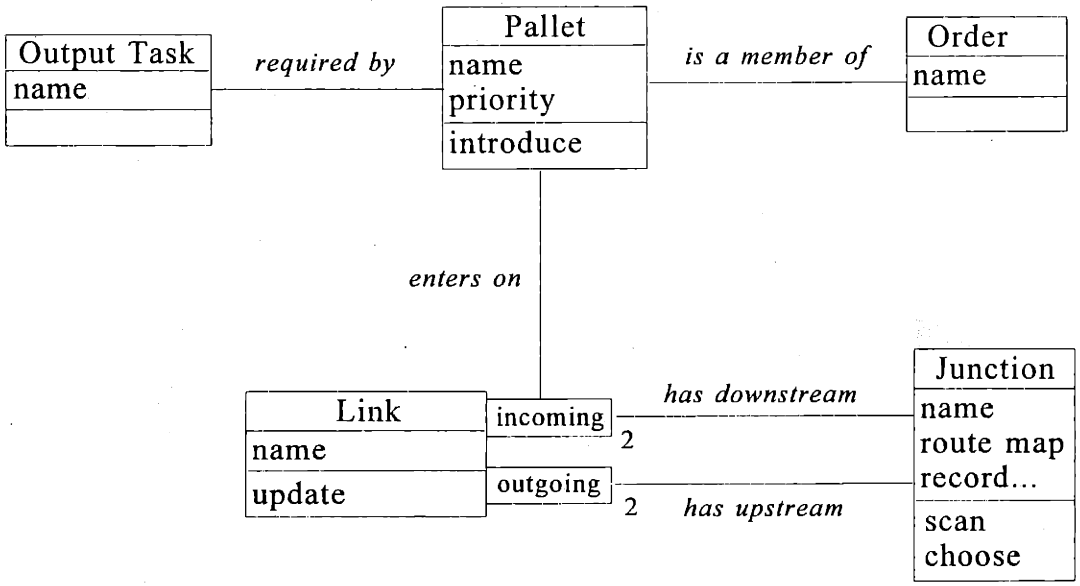
Architecture B, Action Scenario #7: Dynamic Model, Sheet 1



Architecture B, Action Scenario #7: Dynamic Model, Sheet 2



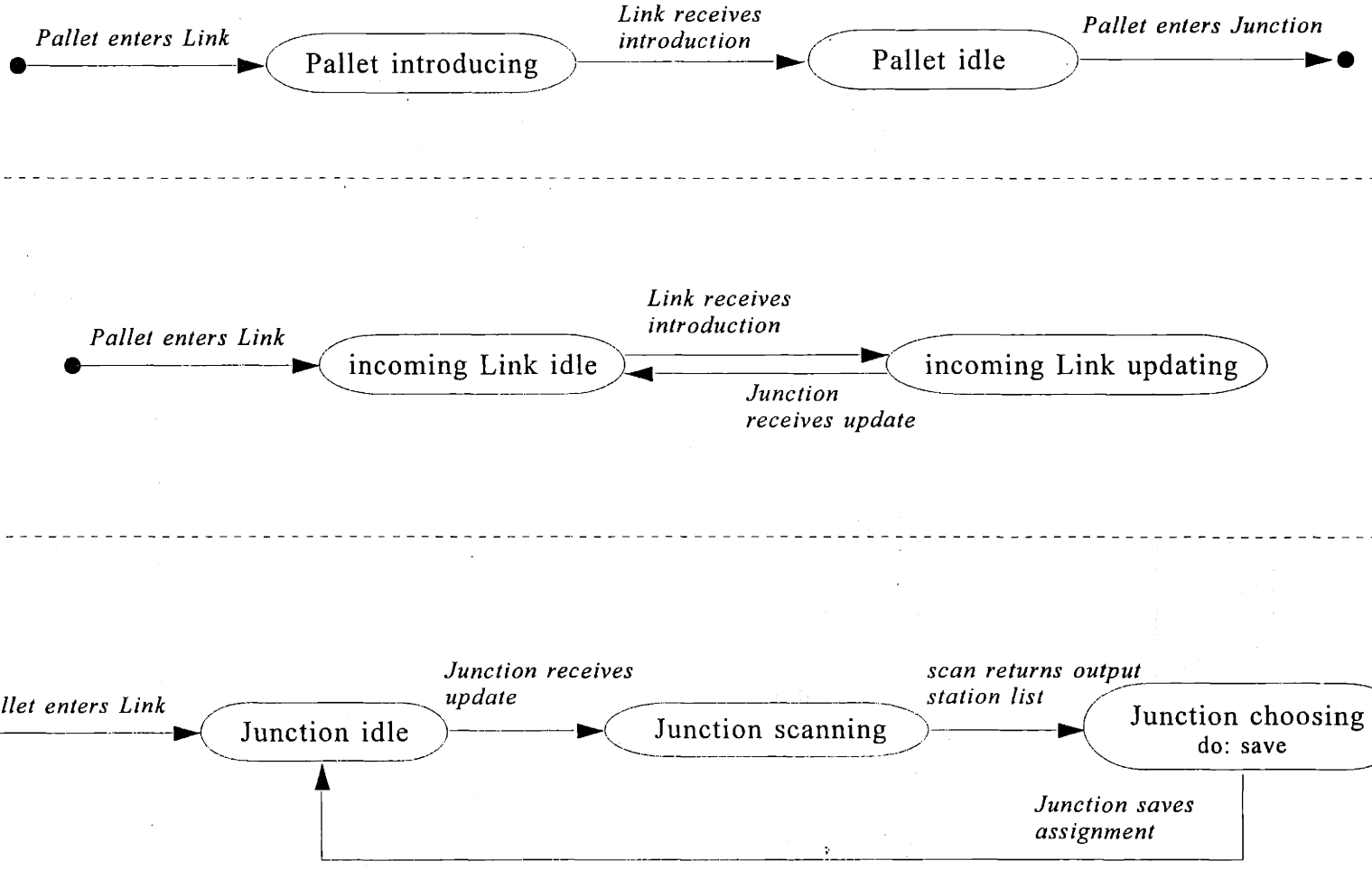
Architecture B, Action Scenario #7: Functional Model



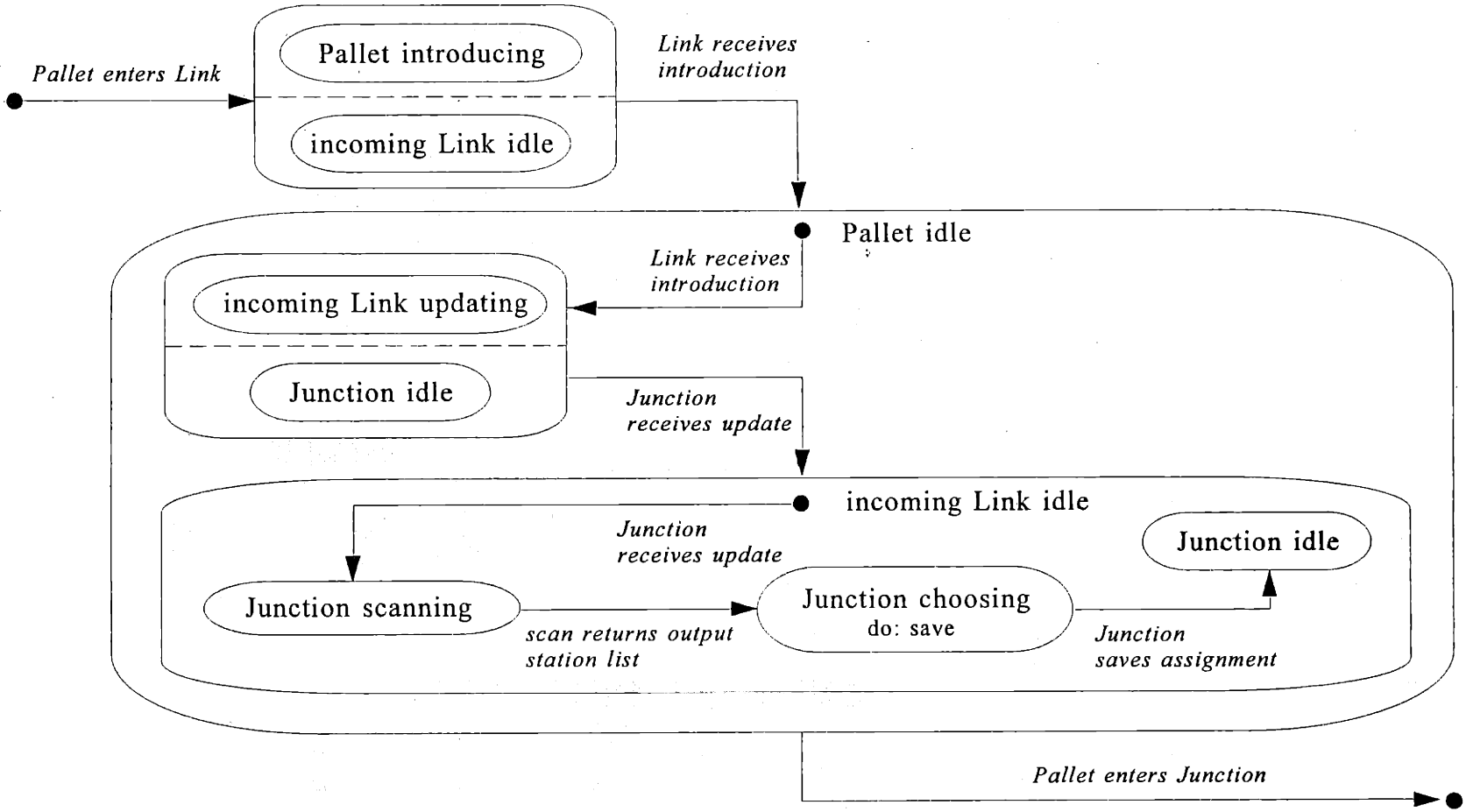
Junction record

1. incoming Link.name
2. Pallet.name
3. Pallet.priority
4. Order.name
5. "Output Station List"
6. outgoing Link.name

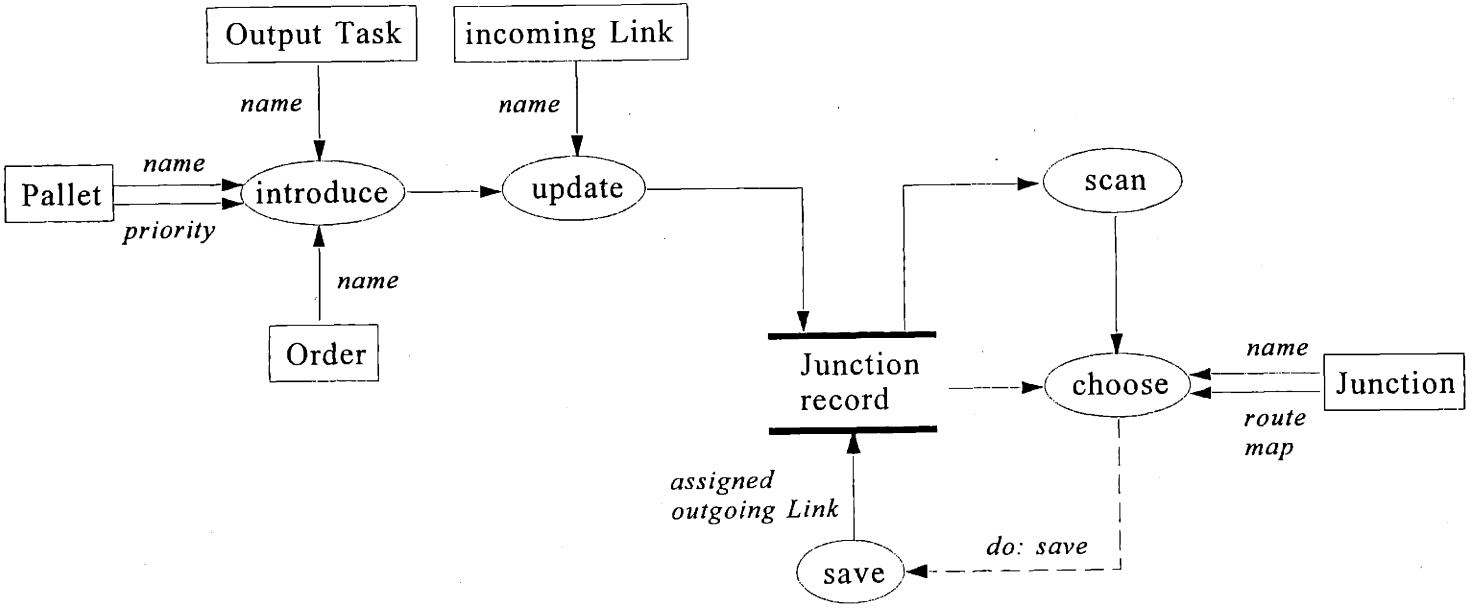
Architecture B, Action Scenario #8: Object Model



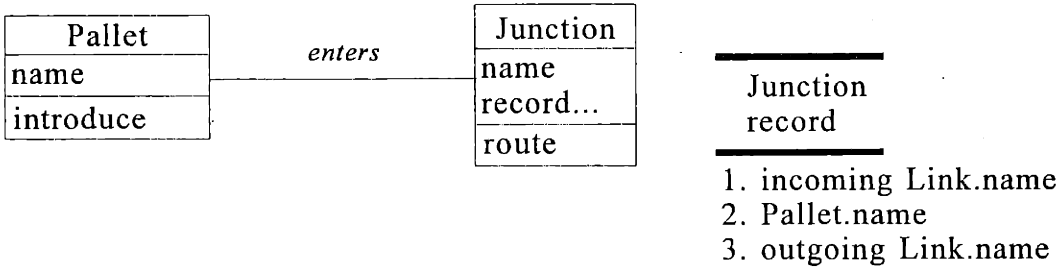
Architecture B, Action Scenario #8: Dynamic Model, Sheet 1



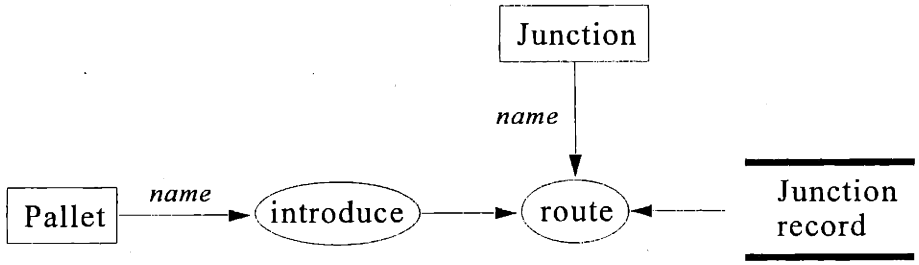
Architecture B, Action Scenario #8: Dynamic Model, Sheet 2



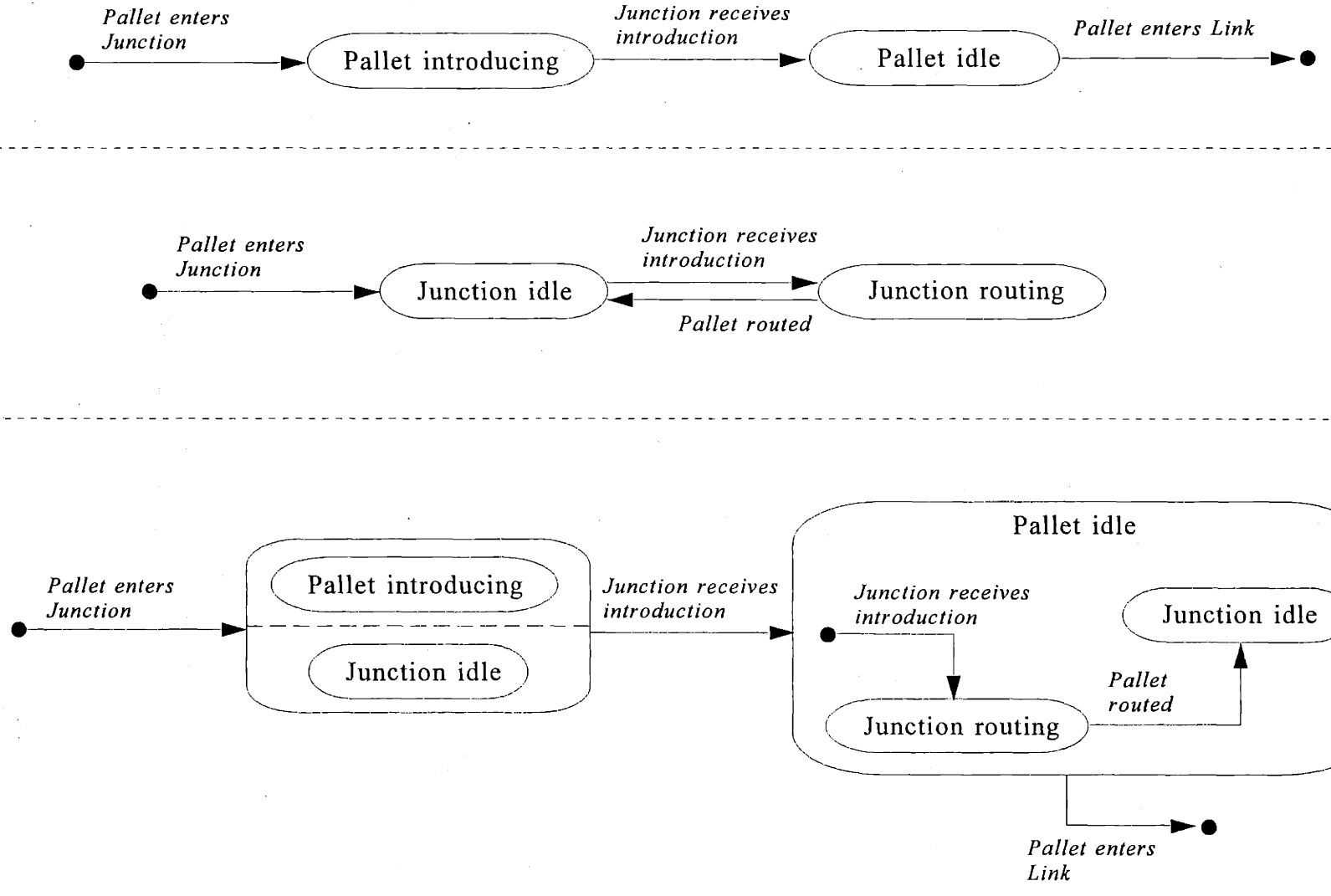
Architecture B, Action Scenario #8: Functional Model



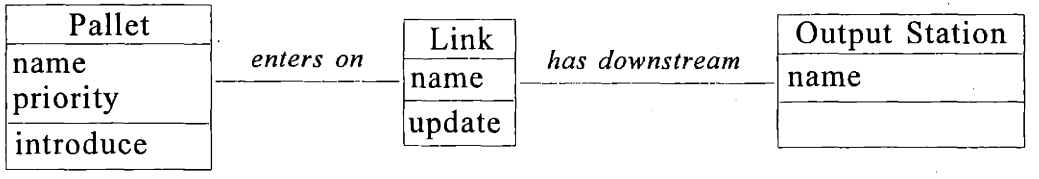
Architecture B, Action Scenario #9: Object Model



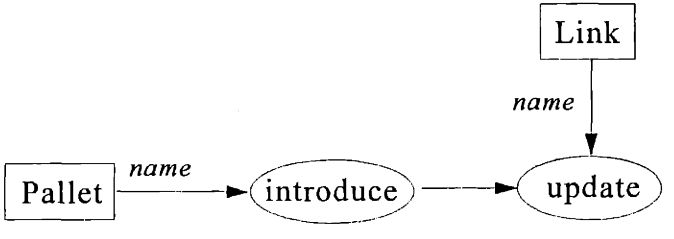
Architecture B, Action Scenario #9: Functional Model



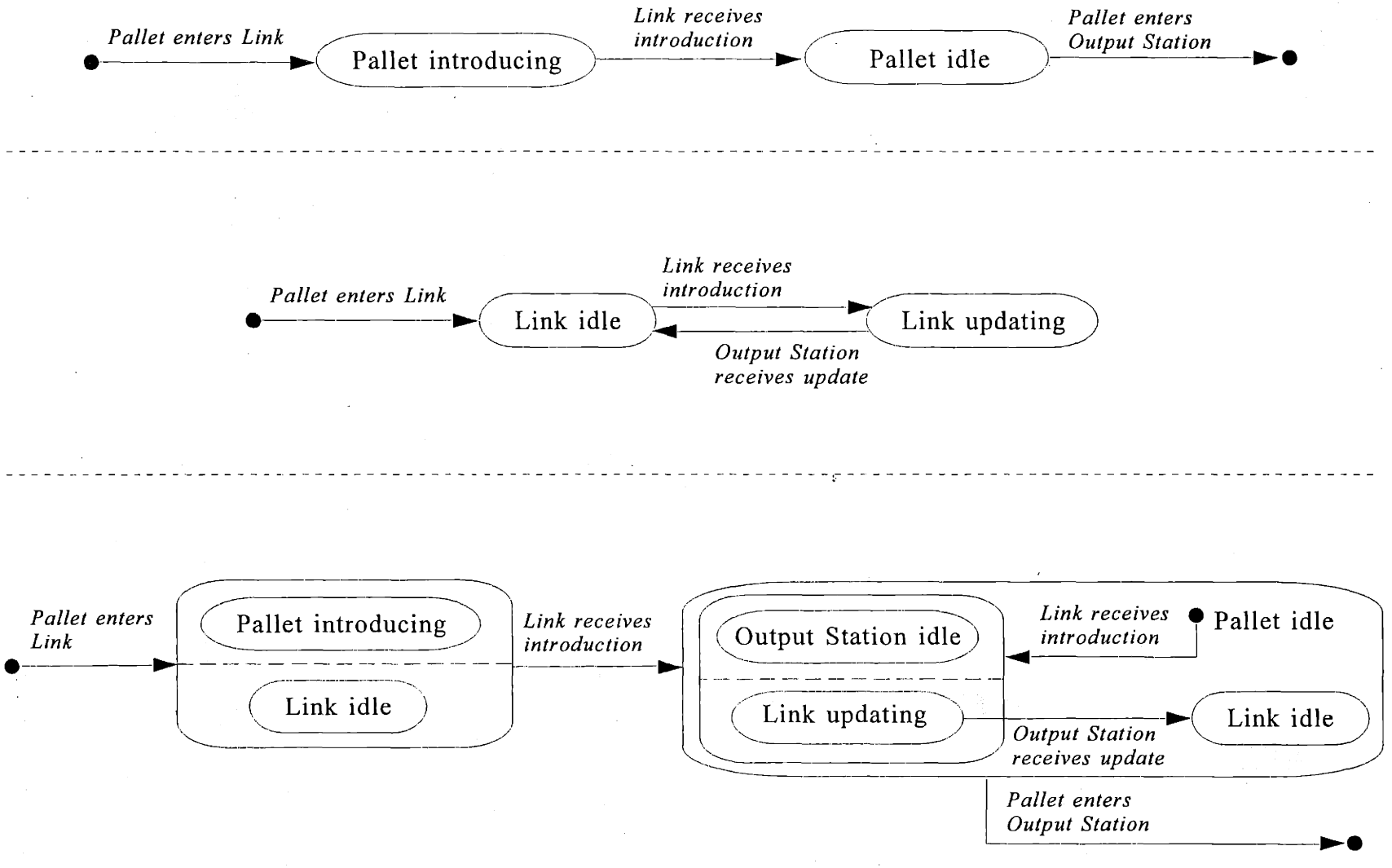
Architecture B, Action Scenario #9: Dynamic Model



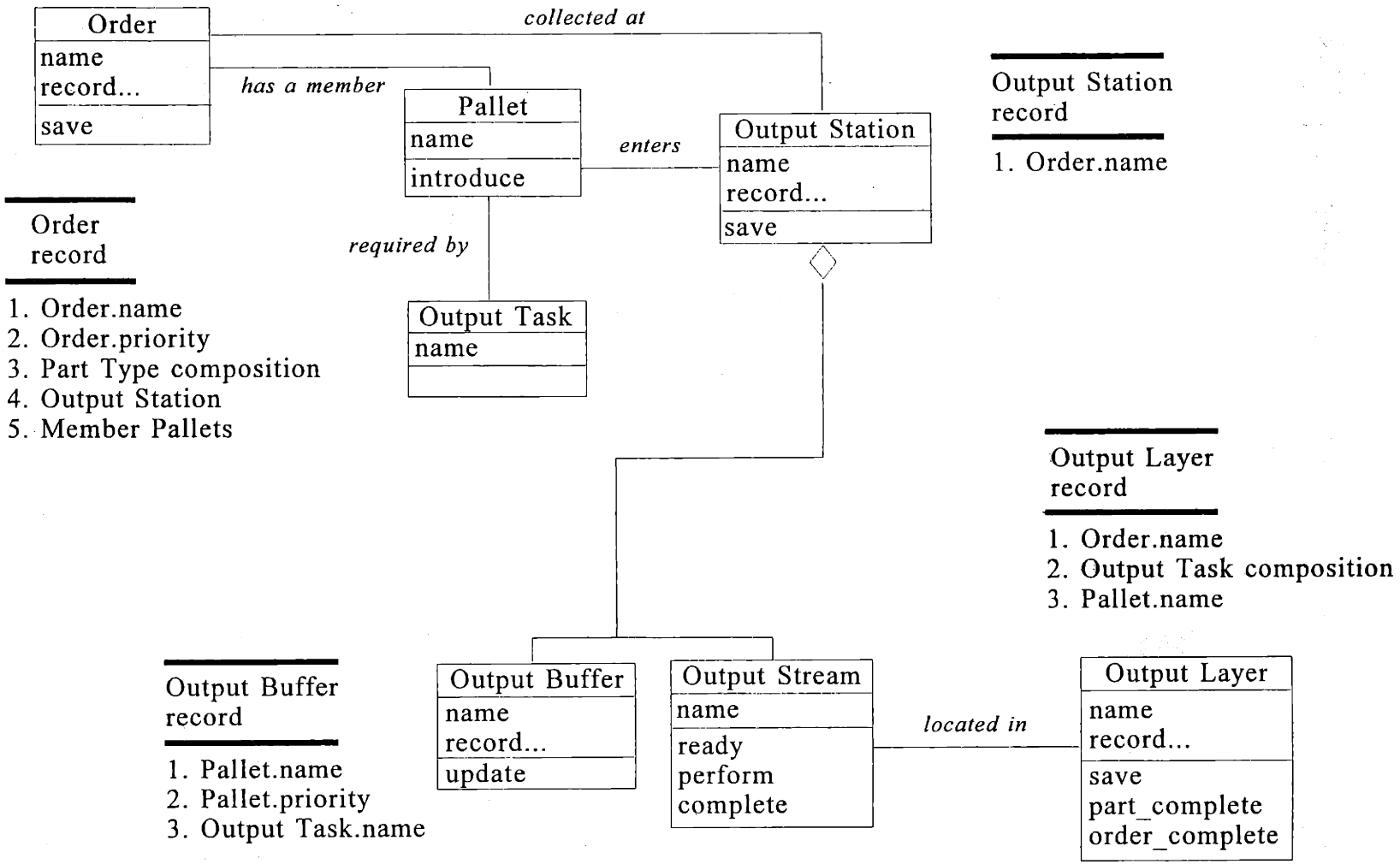
Architecture B, Action Scenario #10: Object Model



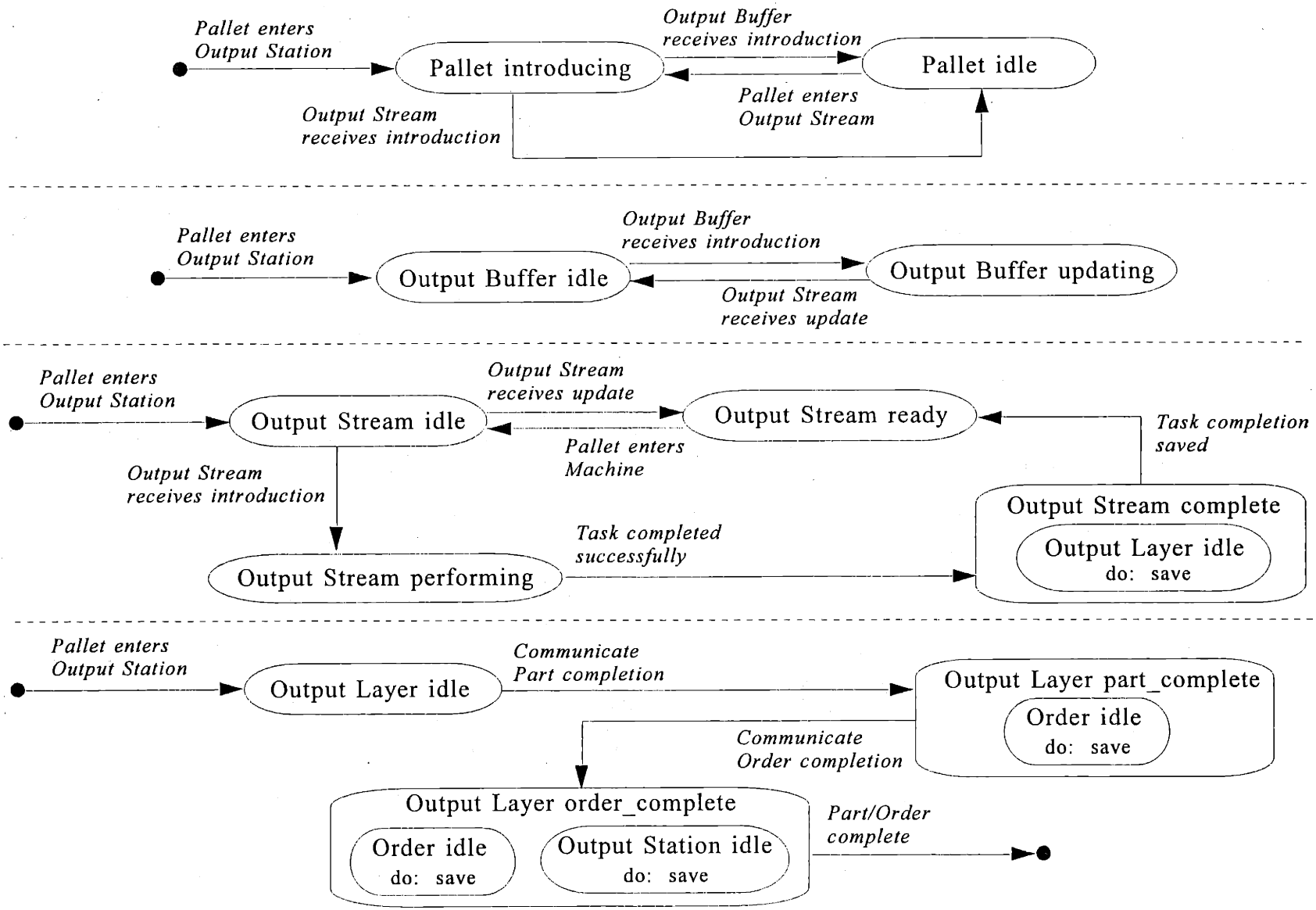
Architecture B, Action Scenario #10: Functional Model



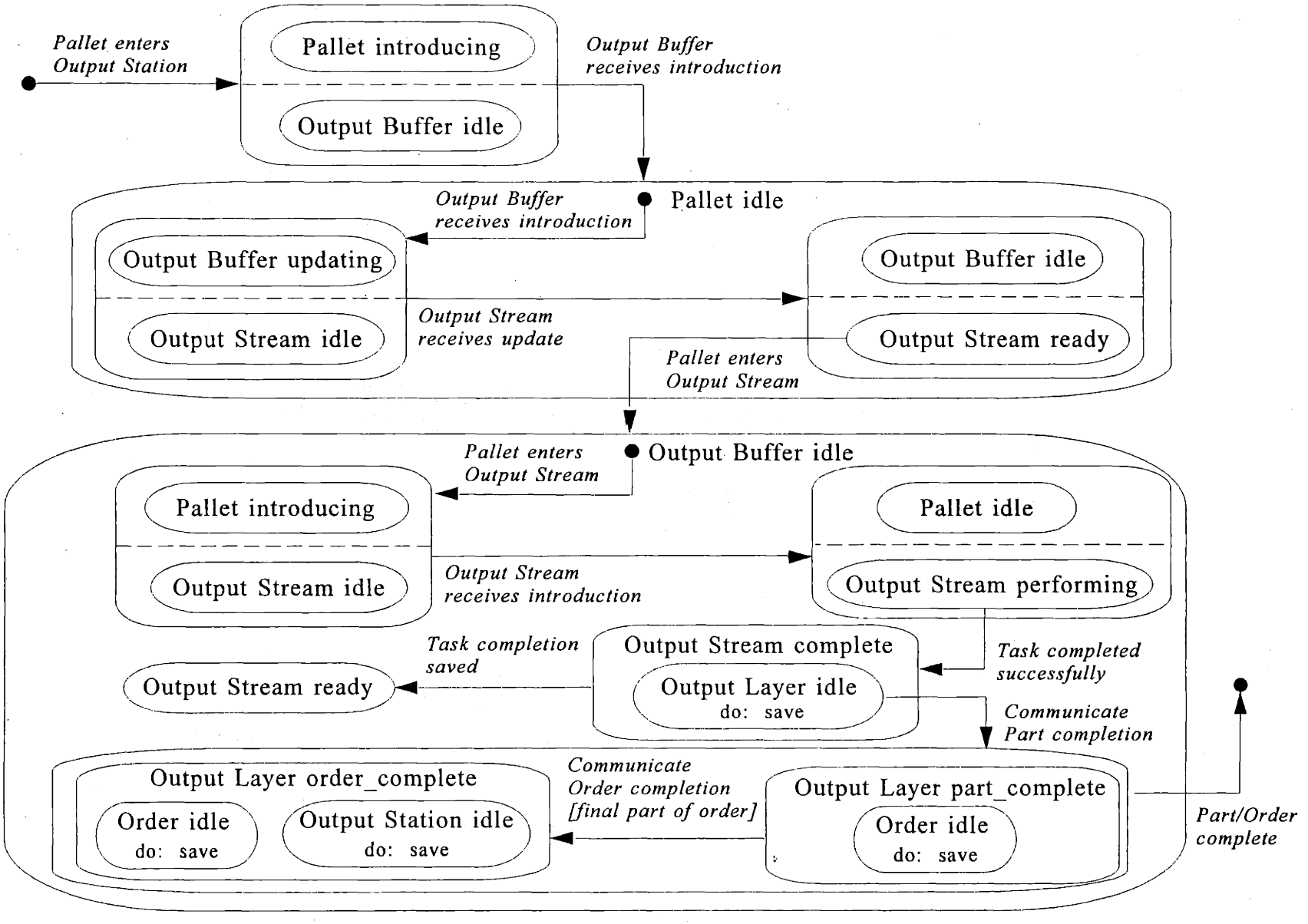
Architecture B, Action Scenario #10: Dynamic Model



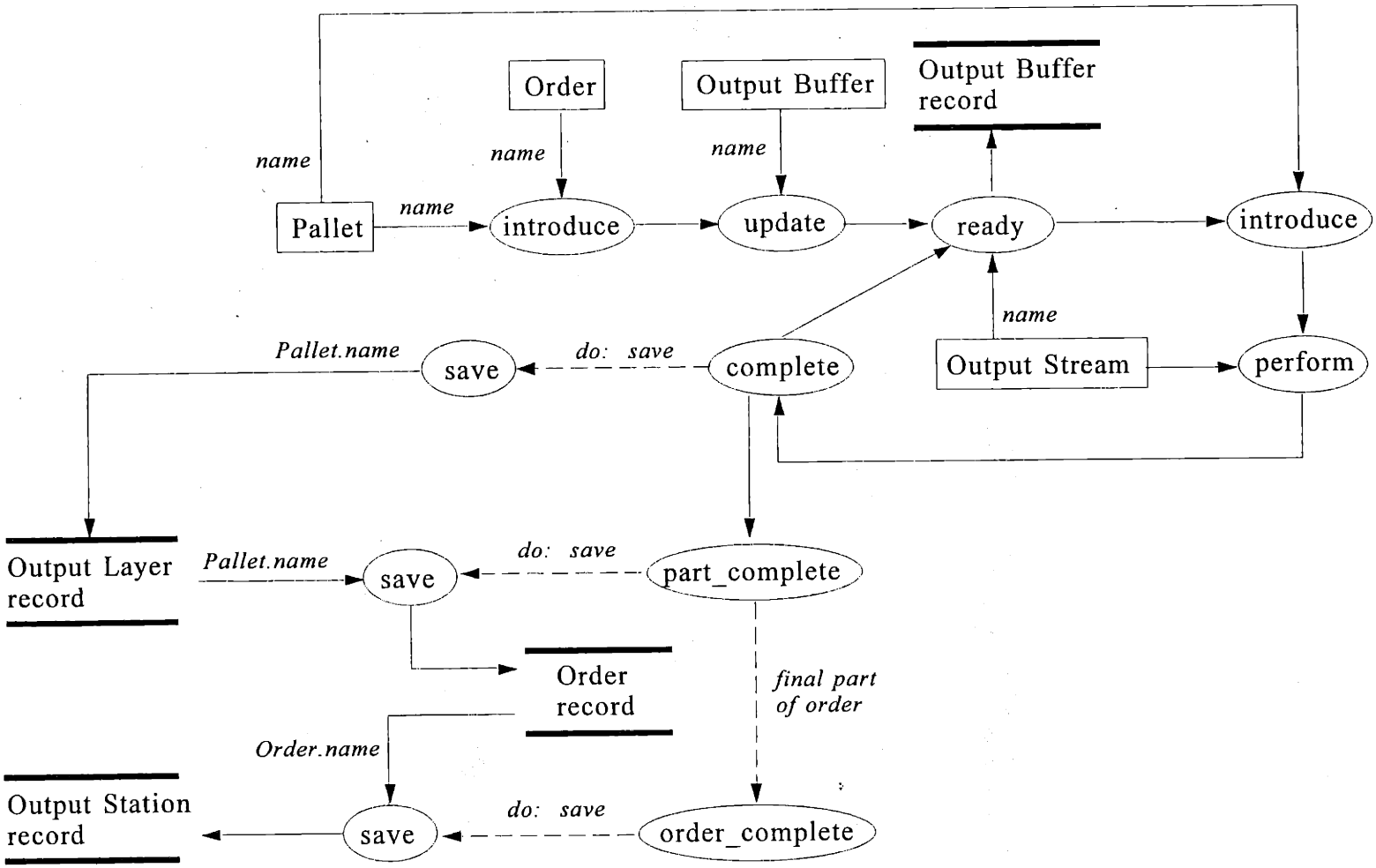
Architecture B, Action Scenario #11: Object Model



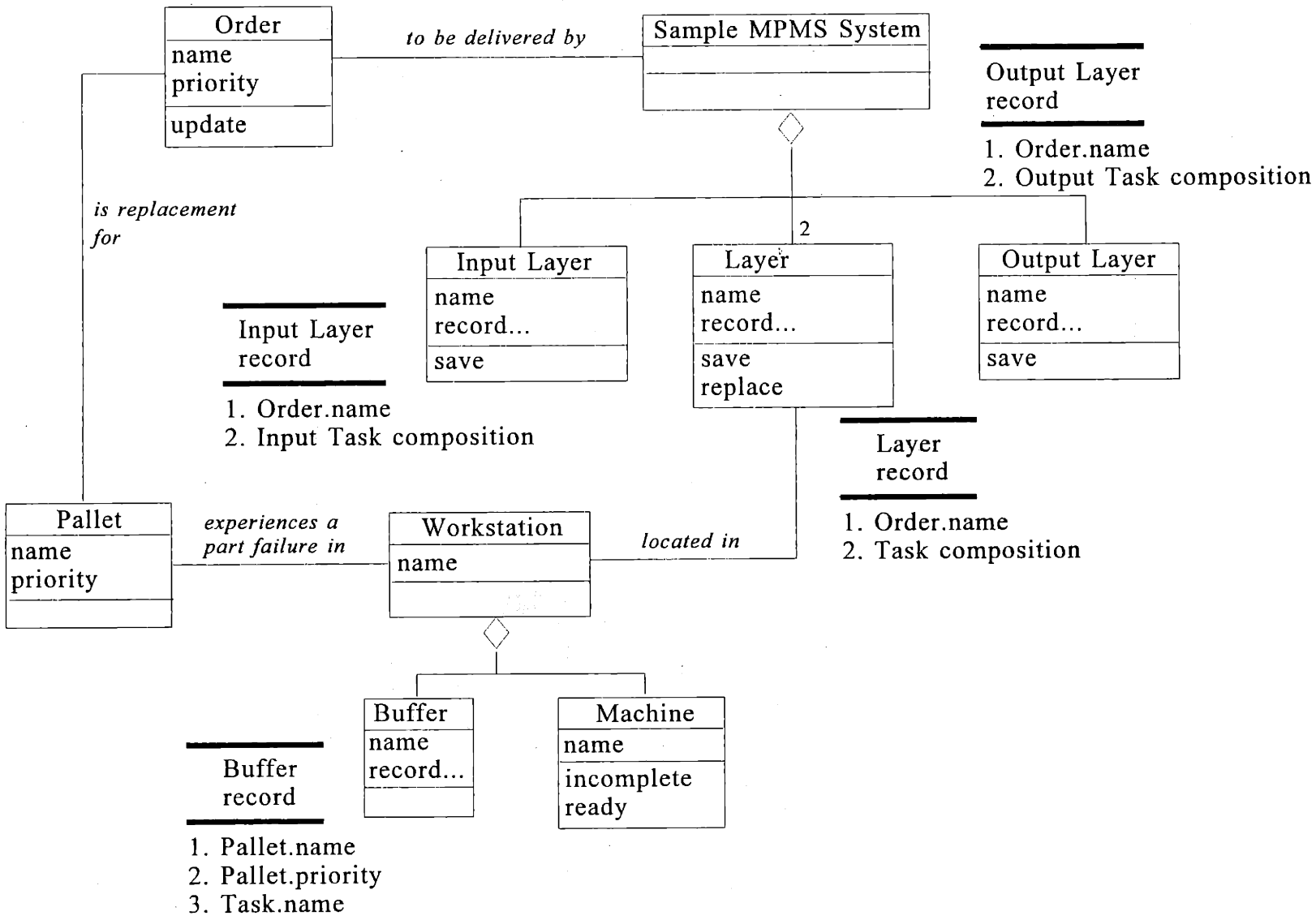
Architecture B, Action Scenario #11: Dynamic Model, Sheet 1



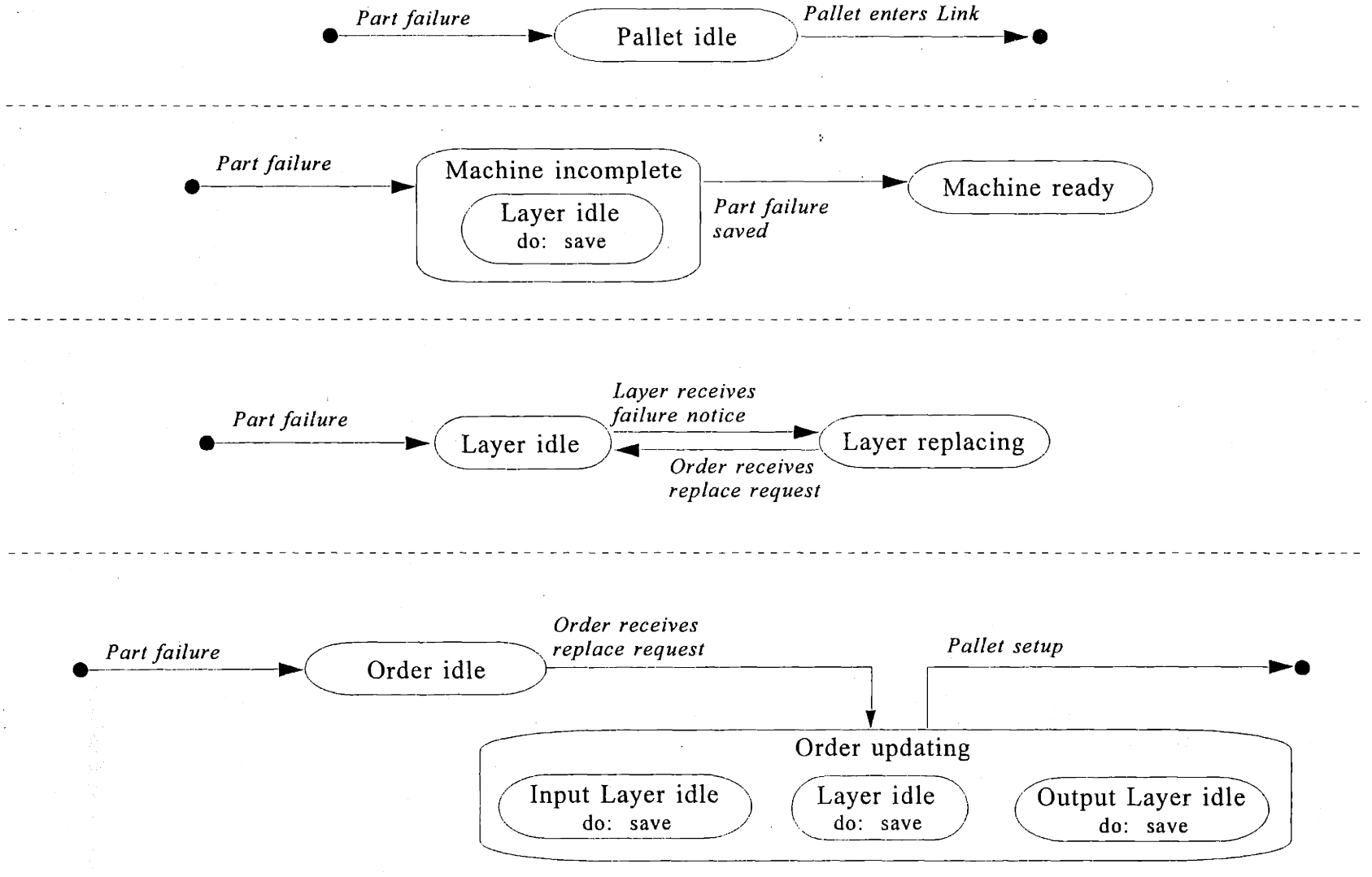
Architecture B, Action Scenario #11: Dynamic Model, Sheet 2



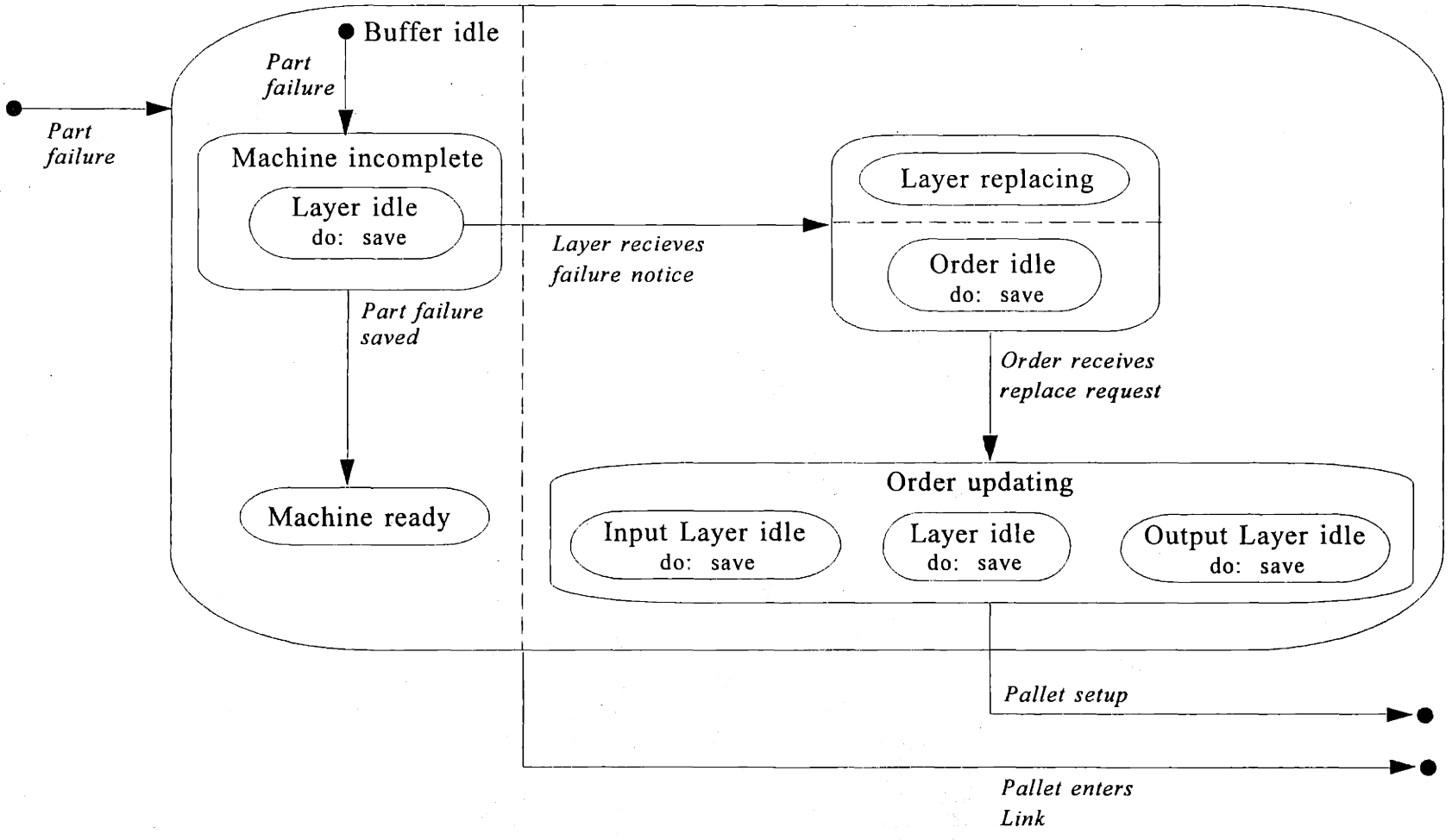
Architecture B, Action Scenario #11: Functional Model



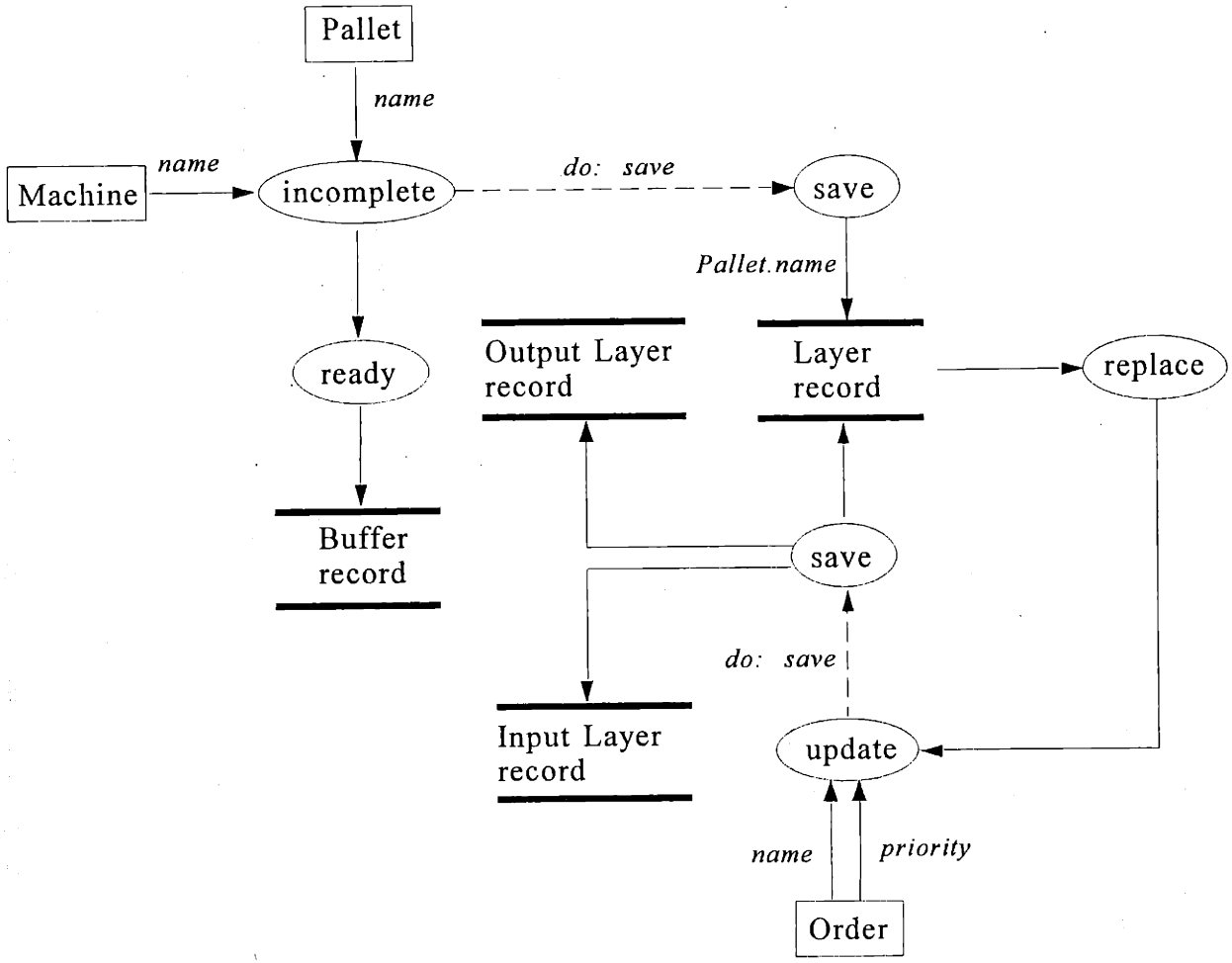
Architecture B, Action Scenario #12: Object Model



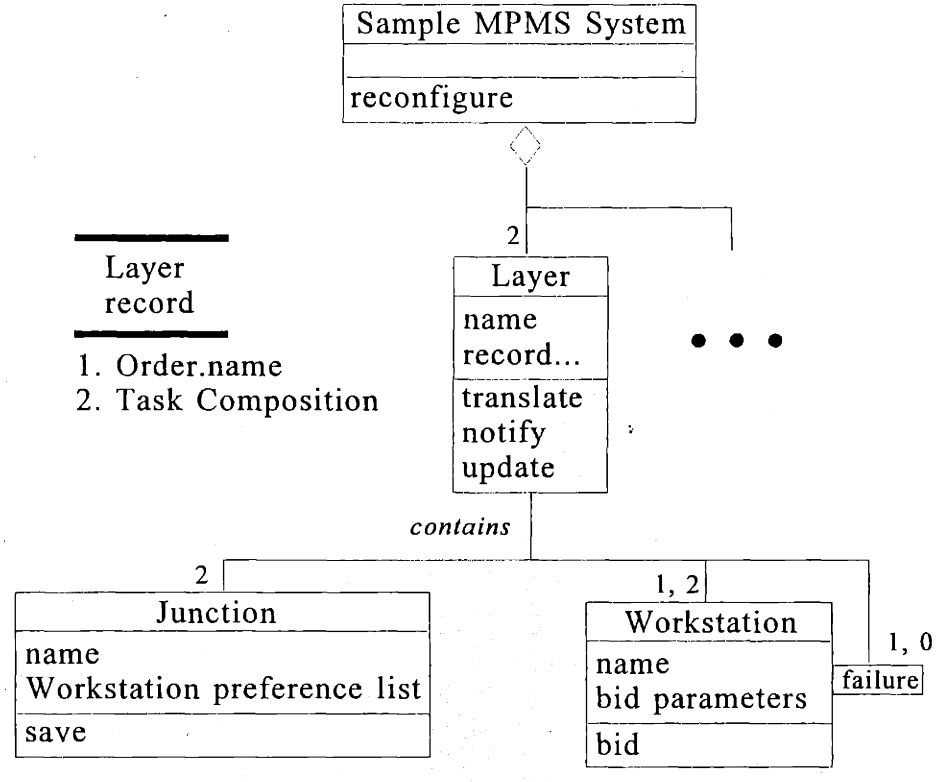
Architecture B, Action Scenario #12: Dynamic Model, Sheet 1



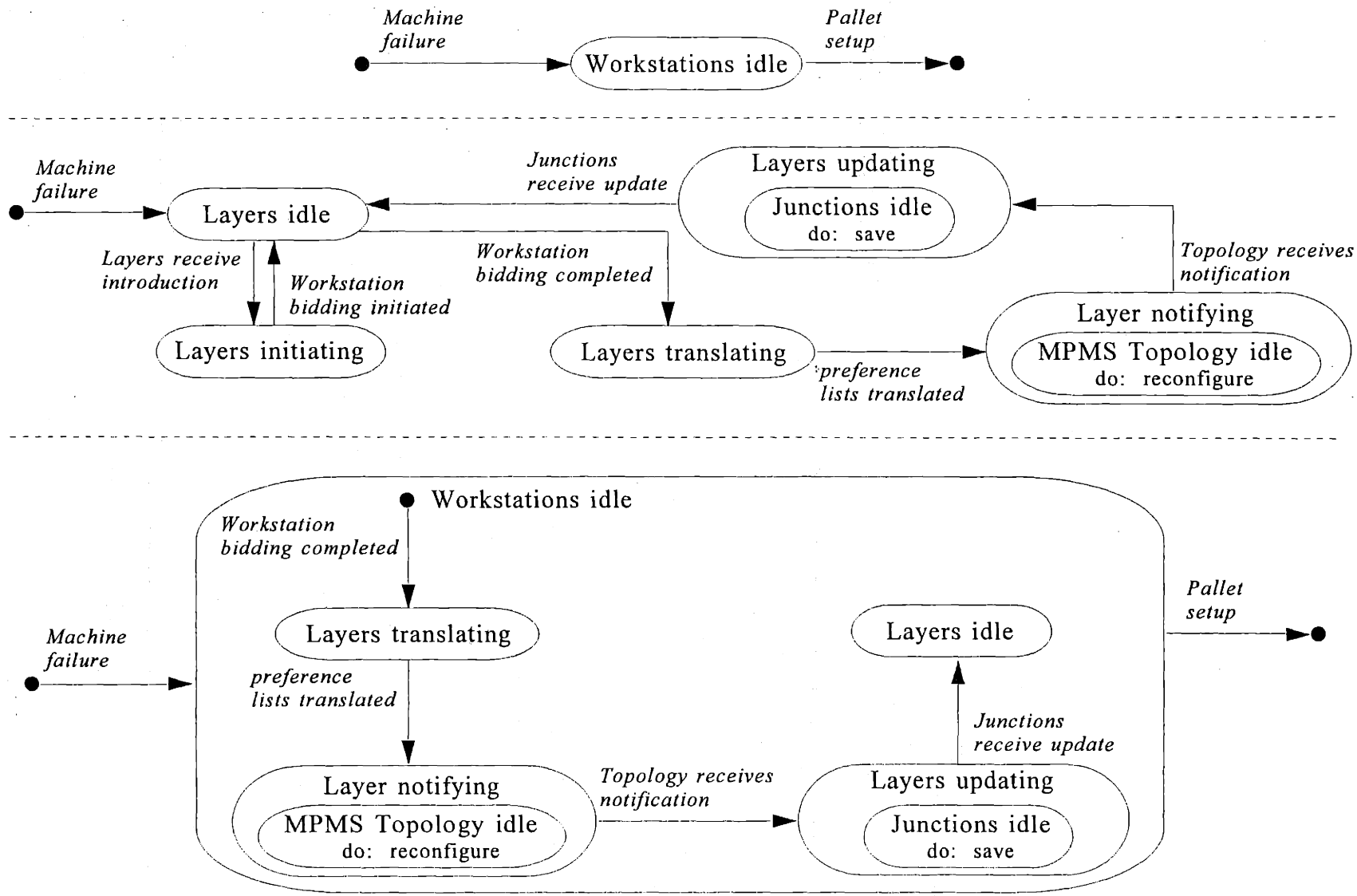
Architecture B, Action Scenario #12: Dynamic Model, Sheet 2



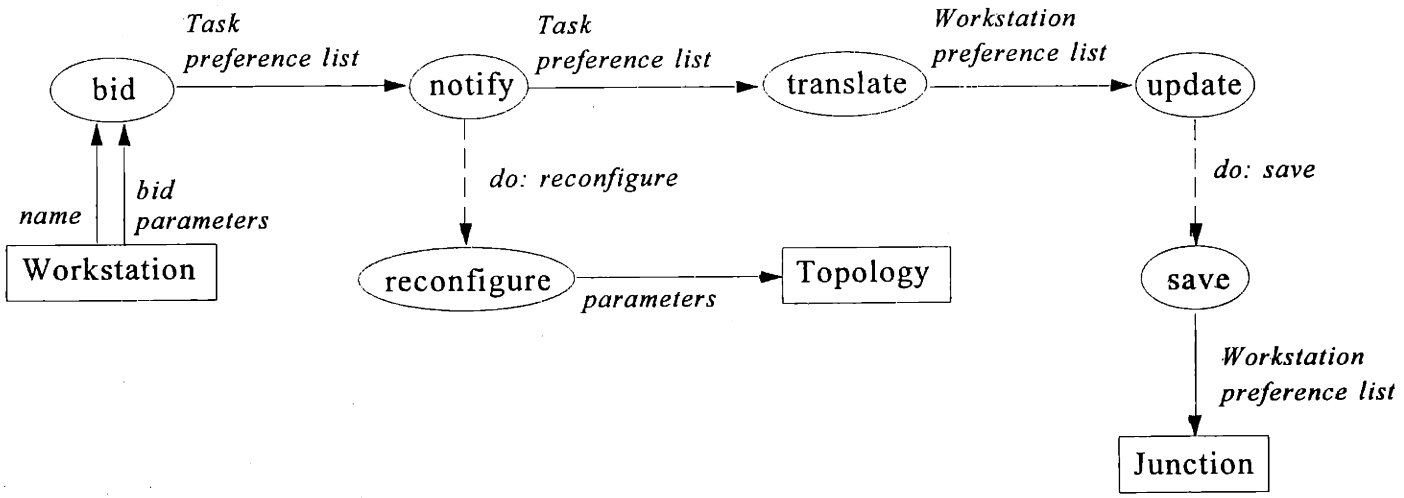
Architecture B, Action Scenario #12: Functional Model



Architecture B, Action Scenario #13: Object Model



Architecture B, Action Scenario #13: Dynamic Model



Architecture B, Action Scenario #13: Functional Model

Bibliography

[1] Theodore J. Williams et al. The role of CIM architectures in flexible manufacturing systems. In Sanjay B. Joshi and Jeffrey S. Smith, editors, *Computer Control of Flexible Manufacturing Systems: Research and Development*, chapter 1, pages 1–30. Chapman & Hall, London, 1994.

[2] Ming Rao, Qun Wang, and Jianzhong Cha. *Integrated Distributed Intelligent Systems in Manufacturing.* Chapman & Hall, London, 1993.

[3] G.R. Meijer and U. Kirchoff. CIM design methodology: Managing the realization of CIM systems. In C. Kooij, P.A. MacConaill, and J. Bastos, editors, *Realising CIM's Industrial Potential*, volume 2 of *Advances in Design and Manufacturing*, pages 16–25. IOS Press, Amsterdam, 1993.

[4] A. Bauer, R. Bowden, et al. *Shop Floor Control Systems: From Design to Implementation.* Chapman & Hall, London, 1994.

[5] Gilles Michel. *Programmable Logic Controllers: Architecture and Applications.* John Wiley & Sons, New York, NY, 1990.

[6] M. Babb. PLC users get some new packaging options. *Control Engineering*, pages 68–71, March 1994.

[7] B. Tinham. So, what now with distributed control? *C & I*, pages 33–38, May 1994.

[8] Anonymous. New BUS links devices on the factory floor. *Mechatronics*, 1994.

[9] A. Chatha. Fieldbus: The foundation for field control systems. *Control Engineering*, pages 77–80, May 1994.

[10] S. Adiga, editor. *Object-Oriented Software for Manufacturing Systems.* Chapman & Hall, London, 1993.

[11] J.K Chaar et al. Efficient and dependable manufacturing - a software perspective. In Joshi and Smith [1], chapter 13, pages 343–378.

[12] Grace Y. Lin and James J. Solberg. Autonomous control for open manufacturing systems. In Joshi and Smith [1], chapter 7, pages 169–206.

[13] Dimitri P. Bertsekas and John N. Tsitsiklis. *Parallel and Distributed Computation: Numerical Methods.* Prentice Hall, Englewood Cliffs, NJ, 1989.

[14] Dimitri P. Bertsekas. *Dynamic Programming and Optimal Control*, volume 1. Athena Scientific, Belmont, MA, 1995.

[15] Dimitri P. Bertsekas. *Dynamic Programming and Optimal Control*, volume 2. Athena Scientific, Belmont, MA, 1995.

[16] Philip Jin-Yi Lin. *Wide Area Optical Backbone Networks.* PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, February 1996.

[17] M. Kate Senehi et al. Hierarchical control architectures from shop level to end effectors. In Joshi and Smith [1], chapter 2, pages 31–62.

[18] Anoop Kumar Malaviya and Gary Alan Bundell. A multi-level architecture for real-time distributed intelligent systems. In R. Zurawski and T.S. Dillon, editors, *IEEE Workshop on Emerging Technologies and Factory Automation: Technology for the Intelligent Factory*, pages 255–260, Melbourne, Australia, August 1992. CRL Publishing Ltd.

[19] Neil A. Duffie and Rex S. Piper. Nonhierarchical control of manufacturing systems. *Journal of Manufacturing Systems*, 5(2):137–139, 1986.

[20] J. Hatvany. Intelligence and cooperation in heterarchical manufacturing systems. In *Proceedings of the 16th CIRP International Seminar on Manufacturing Systems*, pages 1–4, Tokyo, Japan, 1984.

[21] David Ben-Arieh and Eric D. Carley. Qualitative intelligent modeling of manufacturing systems. In Joshi and Smith [1], chapter 10, pages 264–284.

[22] Yves Dallery and Stanley B. Gershwin. Manufacturing flow line systems: a review of models and analytical results. *Queuing Systems: Theory and Applications*, 12:423–494, 1992.

[23] Manjunath Kamath. Recent developments in modeling and performance analysis tools for manufacturing systems. In Joshi and Smith [1], chapter 9, pages 231–263.

[24] James Rumbaugh et al. *Object-Oriented Modeling and Design.* Prentice Hall, Englewood Cliffs, NJ, 1991.

[25] James Martin and James Ordell. *Object-Oriented Methods: A Foundation.* PTR Prentice Hall, Englewood Cliffs, NJ, 1995.

[26] C. Roger Glassey and Sadashiv Adiga. Conceptual design of a software object library for simulation of semiconductor manufacturing systems. *Journal of Object-Oriented Programming*, pages 39–43, Nov/Dec 1989.

[27] Sadashiv Adiga and Milind Gadre. Object-oriented software modeling of a flexible manufacturing system. *Journal of Intelligent and Robotic Systems*, 3:147–165, 1990.

[28] C.R Glassey and S. Adiga. Berkeley library of objects for control and simulation of manufacturing (BLOCS/M). In Lewis J. Pinson and Richard S. Wiener, editors, *Applications of Object-Oriented Programming*, chapter 1, pages 1–27. Addison-Wesley Publishing Co., 1990.

[29] Patric Timmermans et al. On the modular design of shop-floor control systems. In Kooij et al. [3], pages 124–137.

[30] Gabriele Elia and Giuseppe Menga. Object-oriented design of flexible manufacturing systems. In Joshi and Smith [1], chapter 12, pages 315–342.

[31] J.M Hopkins et al. An object-oriented control architecture for flexible manufacturing cells. In Joshi and Smith [1], chapter 16, pages 427–466.

[32] Erich Gamma et al. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Publishing Co., Reading MA, 1995.

[33] Christopher N. Peters. Performance analysis of scheduling policies for a class of flexible automated manufacturing systems through simulation. Master's thesis, Massachusetts Institute of Technology, Cambridge, MA, February 1996.

[34] David Harel. Statecharts: a visual formalism for complex systems. *Science of Computer Programming*, 8:231–274, 1987.

[35] John James et al. The state of computer-aided control system design (CACSD). *IEEE Control Systems Magazine*, 15(2), April 1995.