Device Ecology Workflows with Semantics: Formalizing Automation Scripts for Ubiquitous Computing

Gerry Butler, Seng Loke, and Sea Ling,[‡]

Abstract

We envision collections of (smart) devices in a ubiquitous computing setting working together for the user in the way as specified or programmed. We have been exploring a high-level abstraction for programming interactions among devices based on the workflow paradigm, akin to the automation scripts of the late Michael Dertouzos.[1] In doing so, we recognize the need for a precise description of such an abstraction.

---

*

[†]G. Butler and S. Ling are with Caulfield School of Information Technology, Monash University. S. Loke is with Department of Computer Science and Computer Engineering, La Trobe University.

[1]M. Dertouzos, The Unfinished Revolution: How to Make Technology Work for Us-Instead of the Other Way Around, Collins, 2002.

We have developed a formal model for a device ecology: a collection of communicating devices in a shared environment. Our model defines the communicating processes between a central controller, which we call a *DecoFlow engine*, and the agents contained in the devices. The DecoFlow engine orchestrates the agents' behaviour based on commands specified in a highly concurrent language, *eco*. We demonstrate that an eco procedure ultimately terminates by returning to its initial state. Our model is expressed in π-calculus, as its message-passing semantics and its ability to express agent mobility will be invaluable when we later extend the model.

The model provides semantics for eco procedures, and a foundation for simulation and analysis (within an existing theory, i.e. π-calculus) of how a collection of devices can work together as specified via eco.

Acknowledgements

We thank the Australian Research Council for financial support of this work under grant DP0450092.

1 Introduction

We will be increasingly surrounded by a proliferation of devices (worn on users and embedded into our everyday environments). There is a need and opportunity to program how these devices could work together. Such devices may exist in working relationships with one another in the sense of being

harnessed together towards a user-specified outcome and being situated in the context of the user, forming a *device ecology*. A device ecology is a collection of communicating devices in a common environment. The environment is contained, in the sense that only authorized devices are able to communicate, but its physical size may extend across a living room or across the world.

The devices' *raison d'être* is to serve a *user*. We call the workflow associated with the devices a *Device Ecology Workflow* (or *DecoFlow*, for short.) A central controller, which we call a DecoFlow engine, allows the user to program the collection of devices as if they are a single entity. The DecoFlow engine serves to orchestrate the device collection: it issues pre-programmed commands to devices at specified times and in the required order to achieve the user's goals. Devices may also communicate directly with each other and with the user. The user specifies commands to the DecoFlow engine in an English-like language. Our model supports any language that provides the features described in Section 3. We say that any such language is a member of the *eco* family of languages.

In this paper we develop equations in π-calculus for an ecology of devices controlled by a set of DecoFlow engine instances. In the remainder of this paper, the term *instance* means an instance of a DecoFlow engine. We describe the ecology as if the devices are in a familiar home environment. However, the environment could be an office, a factory, a hospital, a battlefield, or anywhere.

The model we develop forms a foundation for Decoflows, making precise what is often an informal and vague notion of "devices working together". π-

calculus provides a convenient theoretical framework in which Decoflows can be given semantics and studied (as expressions in the calculus). Our model formalizes the analogy (or metaphor) of the workflow (traditionally meaning business process) being applied to describe orchestrated devices working together. The model also provides a foundation for implementing Decoflows based on existing workflow-oriented technologies such as BPEL4WS, as first proposed in [10].

The rest of this paper is organized as follows. Section 2 briefly reviews related work. Section 3 introduces the eco family of languages. Section 4 introduces a simple decoflow, which we use as a running example throughout the paper. Section 5 outlines our formal notation, including the essential features of π-calculus. Section 6 introduces the objects that form the foundation of our model. Section 7 defines the core processes of our model and the interactions between a DecoFlow engine and the agents. Section 8 lists the transitions that occur as a DecoFlow evolves and proves that every sequence of transitions must ultimately terminate. Section 9 concludes with our plans for future work.

2 Related Work

There has been significant work in building the networking and integrative infrastructure for such devices, within the home, the office, and other environments and linking them to the global Internet. For example, UPnP [20], SIDRAH [5] and Jini [13] provide infrastructure for devices to be inter-

connected, find each other, and utilize each other's capabilities.

Embedded Web Servers [2] are able to expose the functionality of devices as Web services. Approaches to modelling and programming such devices for the home have been investigated, where devices have been modelled as software components, collections of objects [1], and Web services [12].

Getting devices connected via Web services is the aim of the proposed Device Profile for Web Services (DPWS)[2] [7, 3] which can be viewed as the next major version of UPnP (UPnP v2) with closer alignment to and taking advantage of standardized Web Service protocols. The various prototypes of DPWS led to embedded devices hosting Web services. In [7] is proposed the Residential Device Controller device which manages a set of devices (three devices mentioned are an alarm, a heater and a shutter), each such device hosting Web services and implementing DPWS. In [3], applications of DPWS enabled devices include cellular network equipment for telecommunications, automotive in-car devices, home appliances, and cross-domain applications (e.g., connecting home and in-car appliances). Hence, the idea of being able to "talk to" and control devices by invoking their Web services is not new and we can expect to see more advanced versions of these in time to come and more of such devices. The gap that this paper fills is, given a collection of such devices, each exposing Web services, how one can utilize collections of these devices in a coordinated manner. The paper proposes eco scripts with a π-calculus semantics for this purpose.

Recent work has developed frameworks for aggregating, composing and

[2]See http://specs.xmlsoap.org/ws/2005/05/devprof/devicesprofile.pdf

building connections among networked devices [16, 9, 4, 15, 8, 21, 19, 11]. However, there has been little work on specifying at a high level of abstraction (and representing this specification explicitly) how such devices would work together at the user-task or application level, and how such work can be managed. Our earlier work in [10] introduced *device ecology workflows* as a metaphor for thinking about how collections of these devices (or devices in a device ecology) can work together to accomplish a purpose. [17] investigates mechanisms to permit a robot to recognize valid commands in spoken sentences that may not be entirely grammatically correct. This forms a building block for the input of device commands.

In [6], UPnP itself was formalized using an abstract state machine model in AsmL. The aim of the formalization was to resolve ambiguities, incompleteness, loose ends, or inconsistencies. Key entities used are agent, communicator, control point and device, and an application is modelled as combinations of these key entities. UPnP communication protocols are formalized in the paper. Our work formalizes device ecology workflows, rather than low-level protocols, providing a semantics for the high-level *eco* family of languages.

3 The *eco* family of languages

Construction of the formal model has been motivated by our work on mobile processes. This has included the design of the DecoFlow engine to control a collection of devices, and a family of languages, *eco*, to allow users to program

the engine. An eco language can have a grammar of any type: regular, context-free, context-sensitive or phrase structured. It is considered that ultimately a phrase structured language will offer the most natural means of expressing commands. However, our work so far has been limited to context-free languages.

An eco procedure consists of *task statements* and *dependency statements* expressed in any language of the eco family. A task statement specifies a particular command to be sent to a particular agent. A dependency statement specifies that a response from a certain task statement s_1 is a prerequisite for the execution of some other task statement s_2. s_1 is said to be an *explicit prerequisite* for s_2. An agent processes at most one command at a time. When an instance attempts to execute a statement referencing a busy agent, the statement is not executed; it remains eligible for execution later.

Eco task statements are executed concurrently subject to explicit prerequisites. An eco language is not an imperative programming language; it does not have sequencing statements, and the lexical ordering of task statements implies nothing about their execution order. An eco procedure can be expressed as a directed acyclic graph, where each node is a task statement and each edge represents an explicit prerequisite. A parser processes an eco procedure to represent it as a DAG, which it passes to an instance of the DecoFlow engine.

The *eco* language family and the parser lie outside the boundary of our formal model. The DAG representing an eco procedure is a given input to our model.

4 Example

We use this example to illustrate our model, comprising a series of commands on devices:

> *make coffee; turn lights dim; wait for lights; show news on tv; wait for tv; turn lights bright; wait for tv; make coffee.*

This example is expressed in a particular member of the eco family of languages which, for the purposes of this example, we call *ecoA*. Figure 1 shows the task statements in this procedure and their dependencies. In ecoA, the *wait for X* dependency statement specifies that the most recent lexically preceding task statement referencing agent X is an explicit prerequisite for the task statement that lexically follows *wait for X*. Although ecoA is not sufficiently expressive to specify a dependency between every pair of task statements in an arbitrary procedure, it is adequate for our example.

In this example, the term *coffee* refers to the brewing agent in the coffee machine. The coffee machine can make only a single style of coffee; if it could make multiple styles of coffee, then the coffee style would need to be an argument, for example *make turkish coffee*.

There is a tradeoff of expressiveness and simplicity in designing *eco*. The language should not be too complex, as users will not be able to utilize it, and the language should not be too simple, as users might be too restricted in what they can do with the devices. We think that *eco* is simple enough for users to write expressions involving up to four commands, but

more technically inclined users will be able to program more complex expressions. Technicians can program a library of complex expressions for a given home, each such expression labelled with a simple command-phrase for use by non-technical end-users. Other user interface designs can be employed; for example, speech recognized commands can be employed on the user interface end with such speech mapped to *eco* commands.

Moreover, we envision that different types of environments (e.g., factory, home, network equipment, car) might each utilize a specialized set of commands or a specialized language, a member of *eco*, tailored to the particular environment with suitable commands for the devices typically found in such an environment or tasks typically carried out in the environment. However, similar commands in different environments need to be resolved into its constituent Web service calls to the particular devices found in that environment. For example, a command such as `turn on the lights` can be issued in different rooms, but will result in a different set of lights being turned on each time the command is issued in a different room; this is the application of the linguistic notion of the *efficiency of language* where similar commands can be effectively reused between different environments, and certainly more so, between different environments of the same type (e.g., home environments), than between environments of different types (e.g., reuse of command-phrases between a home environment and a factory environment might be expected to be less). We also note that there is opportunity to develop dialects of *eco* not just for a type of environment but also for a class of devices - for example, there could be a dialect of *eco* for typical commands

on a television or for display devices. We do not discuss development of *eco* dialects extensively in this paper but note the generality and applicability of our proposed semantics for such purposes.

5 Formalism

π-calculus is the calculus of communicating, concurrent, mobile processes ([14] and [18]). In this paper we use only a subset of π-calculus. We have chosen π-calculus because of its message-passing semantics, and because its ability to express agent mobility will be invaluable when we later extend our model. We use the following notations of π-calculus and adapt them to meet our needs. In the following, P, Q and R are processes, a is a communication channel and n is command or control.

$$P \stackrel{def}{=} \bar{a}n \cdot Q \tag{1}$$

describes an action in which P sends n via a and evolves to Q.

$$P \stackrel{def}{=} (x)a \cdot Q \tag{2}$$

describes an action in which P receives a name via a, evolves to Q, and substitutes the received name for all occurrences of x in Q.

$$P \stackrel{def}{=} Q + R \tag{3}$$

describes an action in which either Q or R occurs and the other is voided.

$$P \stackrel{def}{=} \tau \cdot Q \tag{4}$$

describes an action in which P silently evolves to Q.

$$P \stackrel{def}{=} [x = y]\tau \cdot Q \tag{5}$$

describes an action in which P silently evolves to Q if $x = y$ and does nothing otherwise.

$$P \stackrel{def}{=} Q \Big| R \tag{6}$$

describes an action in which Q and R execute concurrently and communicate via shared names. Σ notation is a shorthand for a series of terms separated by $+$, and Π is a shorthand for a series of terms separated by $|$.

Transitions are represented as follows.

$$P \xrightarrow{an} Q \tag{7}$$

describes a transition from P to Q by receiving n on a.

$$P \xrightarrow{\bar{a}n} Q \tag{8}$$

describes a transition from P to Q by sending n on a.

$$P \xrightarrow{\tau} Q \tag{9}$$

describes a silent transition from P to Q.

We also use the following notation. If X is a set, then $x\colon X$ defines the symbol x as a variable ranging over members of X; $\tilde{X}$ denotes the complement of X with respect to some superset of X, where the superset is usually the set of all statements associated with a given eco procedure; $card(X)$ denotes the cardinality of X.

When a symbol denoting an atomic entity has more than one character, the symbol is underlined; for example: the control $\underline{start}$ and the response $\underline{ok}$.

6 Foundations

6.1 Instances, agents and processes

In the real world there may be multiple instances. However, our model is constructed in terms of a single instance, denoted by D. An instance has a channel, denoted by b, through which it can receive controls and send responses. Controls and responses are defined in Section 6.2.

An instance is initially $free$; it is *busy* while processing a procedure. and, when it finishes, it sends a response and becomes $free$. We distinguish between an instance and the process it contains. In π-calculus a process becomes a different process when it changes state. Hence, in our model an instance is associated with a sequence of processes: the instance's initial process evolves as its state changes.

Each device may contain multiple agents and each agent can perform one operation at a time. Agents are $free$ until they receive a command; they are *busy* while processing the command and become $free$ after sending a response. If it is necessary for a physical device to perform multiple operations concurrently, then our model permits it to contain multiple concurrent agents; the agents may be performing different tasks, or several agents may

be performing the same task. For example, the TV may have one viewing agent (assuming it can show either the news or the sport, but not both at the same time), and one volume control agent. Both the viewing agent and the volume control agent may operate concurrently. The coffee maker may have two brewing agents and one grinding agent, assuming it can brew two coffees and grind the beans for another concurrently. Our model views a collection of agent interfaces, with no regard to how the agents are grouped to form physical devices. The agents forming a physical device may have internal connections and user interactions unknown to the instances. For example, a brewing agent may internally request the grinding agent to provide coffee, and the user may request the brewing agent to pour previously brewed coffee. We distinguish between an agent and the process it contains. Like an instance, an agent contains a sequence of processes that evolve as the agent changes state.

A process X is pending on a process Y when it has sent a command to Y and it has not yet received a response; X may continue processing commands.

The agents forming each physical device are assumed to know how to deal with nonsense commands. For example, assuming the drapes have only one agent, that can accept either an *open* or *close* command, an instance guarantees that it will not issue a *close* command while a previous *open* or *close* command is in progress. However, if the drapes receive a *close* command when they are already closed, then we assume that the *close* command knows whether to respond $\underline{fail}$, or ignore the command and respond $\underline{ok}$.

Each agent has a channel through which it accepts one command or

control at a time and sends responses. Every command eventually ends by responding either $\underline{ok}$ or $\underline{fail}$ through its channel.

In the following, *index set* means a set of consecutive integers beginning at 1. Let P be a finite set of agents. Let I be an index set for P. We denote members of P by P_i, $i \in I$. Agent P_i contains an initial process which we also denote by P_i. The context determines whether we are referring to agent P_i or process P_i. If there is any doubt we explicitly refer to the agent or its initial process, as in the previous sentence. Process P_i represents the initial state of agent P_i, in which it is $free$. Process P_i evolves through a sequence of subsequent processes, which we denote by parenthesized subscripts of the form $P_{i_{(W,R,c)}}$. Section 7.2 defines the agent states.

In our example, $P = \{coffee, lights, tv\}$. In more detail, $I = \{1, 2, 3\}$, and $P = \{P_1, P_2, P_3\}$ where $P_1 = coffee$, $P_2 = lights$, $P_3 = tv$.

We define a *command* as an operation-argument pair. Let O be a finite set of operation names. An operation name need be unique only within each agent. Let G be a finite set of argument values. G includes the value $nullArg$.

Definition 1 (Command) *A* command *is a member of the set* $\{(o\colon O, g\colon G) | g \text{ is an argument of } o\}$.

We denote the set of commands by C. Let J be an index set for C. We denote members of C by C_j, $j \in J$. Although arguments play little part in our model, they are included because they occur in the eco language family.

In our example, the set of operation names O is $\{make, turn, show\}$, the set of argument values G is $\{nullArg, dim, news, bright\}$ and the set of com-

mands C is $\{(make, nullArg), (turn, dim), (show, news), (turn, bright)\}$. J, the index set, is $\{1, 2, 3, 4\}$. C can be written $C = \{C_1, C_2, C_3, C_4\}$, where C_1 is $(make, nullArg)$, C_2 is $(turn, dim)$, C_3 is $(show, news)$ and C_4 is $(turn, bright)$.

We define a *task* as a particular command sent to a particular agent.

Definition 2 (Task) *A* task *is a member of the set* $\{(p\colon P, c\colon C) | c$ *is a valid command for* $p\}$.

We denote the set of tasks by T. We denote the member $(P_i, C_j) \in T$ by T_{ij}.

In our example, $T = \{(coffee, (make, nullArg)), (lights, (turn, dim)), (tv, (show, news)), (lights, (turn, bright))\}$, or $T = \{T_{11}, T_{22}, T_{33}, T_{24}\}$ where $T_{11} = (coffee, (make, nullArg))$, $T_{22} = (lights, (turn, dim))$, $T_{33} = (tv, (show, news))$, $T_{24} = (lights, (turn, bright))$.

Recall that an eco task statement specifies a particular command sent to a particular agent (Section 3). A task statement can only specify a command that is valid for the agent to which it is sent. Hence, each task statement is associated with some task. Multiple task statements in an eco procedure may be associated with the same task. For example, the task *make coffee* may occur more than once; each occurrence is a distinct task statement. Each task statement is associated with an ordinal number according to its order in its procedure's lexical sequence of task statements. For a given procedure, let L be the set of ordinals. Since each procedure has a finite number of task statements L is finite. A *statement* is the task-ordinal pair derived from a given eco task statement.

Definition 3 (Statement) *A* statement *is a member of* $\{(t\colon T, l\colon L) | t's$ *associated task statement has ordinal* $l\}$.

We denote the statement set by S. By definition, S is finite. We denote the member $(T_{ij}, l) \in S$ by S_{ijl}.

Note that the term **task statement** refers to a language construct in any language of the eco family, while the terms **task** and **statement**, used separately, refer to objects of our model. In our example procedure, the **task statement** *show news on tv* corresponds to the **statement** $((tv, (show, news)), 3)$, which references the **task** $(tv, (show, news))$.

In our example,

$$\begin{aligned} S = \{&((coffee, (make, nullArg)), 1), \\ &((lights, (turn, dim)), 2), \\ &((tv, (show, news)), 3), \\ &((lights, (turn, bright)), 4), \\ &((coffee, (make, nullArg)), 5)\}. \end{aligned}$$

Here, the set of ordinals L is { 1, 2, 3, 4, 5 }, and S can be written

$$S = \{S_{111}, S_{222}, S_{333}, S_{244}, S_{115}\}, \tag{10}$$

where $S_{111} = ((coffee, (make, nullArg)), 1)$, $S_{222} = ((lights, (turn, dim)), 2)$, $S_{333} = ((tv, (show, news)), 3)$, $S_{244} = ((lights, (turn, bright)), 4)$, $S_{115} = ((coffee, (make, nullArg)), 5)$.

We define a relation, *prerequisite*, on S (Section 3).

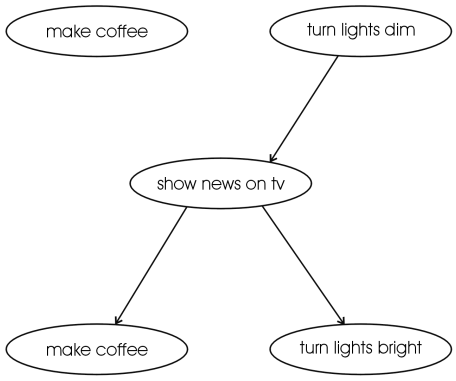

Figure 1: Example Statement Net

Definition 4 (Prerequisite) *A* prerequisite *is a member of* $\{(s_1\colon S, s_2\colon S) |$ s_1 *is an explicit prerequisite for* $s_2\}$.

We denote the prerequisite relation by Δ. By definition, Δ is irreflexive. Δ^+ denotes the transitive closure of Δ. Eco semantics constrain Δ such that there are no circular prerequisites.

Constraint 1 (Circular prerequisite) $(s_1\colon S, s_2\colon S) \in \Delta^+ \implies (s_2, s_1) \notin \Delta$.

Figure 1 illustrates the prerequisite relation in our example; Here, $\Delta = \{(S_{222}, S_{333}), (S_{333}, S_{244}), (S_{333}, S_{115})\}$.

We define the *postset* of a statement as the set of statements for which it is a prerequisite.

Definition 5 (Postset) *The* postset *of a statement* $S_{ijl} \in S$ *is the set* $\{s\colon S | (S_{ijl}, s) \in \Delta\}$.

We denote the postset of S_{ijl} by $S_{ijl}^{\bullet}$. Note that the postset operator $^{\bullet}$ operates on a statement S_{ijl} and produces a set of statements.

We define the *statement net* of an eco procedure as the graph formed from its statement set S and the prerequisite relation Δ.

Definition 6 (Statement net) *The* statement net *of an eco procedure having a statement set* S *and a prerequisite relation* Δ *is the graph* (S, Δ).

We denote a statement net by $\Omega = (S, \Delta)$. Ω is an input to our model.

Theorem 1 Ω *is a directed acyclic graph.*

Proof: By definition, Ω *is a directed graph. We need to show that* Ω *has no cycles. Assume that* Ω *has a cycle and that* s *is a node in the cycle. Then there is a node* s' *in the cycle such that* (s, s') *is an edge of* Ω*, that is,* $(s, s') \in \Delta$*. Since* s *and* s' *are in a cycle, there is a path from* s' *to* s*. From transitivity,* $(s', s) \in \Delta^{+}$*, which contradicts the circular prerequisite constraint (Constraint 1). Hence,* Ω *does not have a cycle.* ■

Ω is the directed acyclic graph mentioned in Section 3.

6.2 Controls, states and channels

Let $control = \{\underline{start}, \underline{stop}\}$ be a set of *controls*, and $response = \{\underline{ok}, \underline{fail}\}$ be a set of *responses*. A control indicates that a statement net is to be started or stopped, or a command is to be stopped. Responses report the outcome of a statement net or a command. We distinguish between *controls* and *commands*: controls are members of the set $control$; commands are members of the set C. Controls, commands and responses may be sent on the same channels. Responses flow in the opposite direction from controls and commands.

We define three subsets of statements: the *pending* set, the *responded* set, and the *eligible* set. A pending statement is currently being processed by an agent. A responded statement has completed processing. An eligible statement is available for processing, that is, it is not pending, it has not responded, and it is not in the postset of any statement that has not responded. Note that S is not necessarily the union of these subsets since, depending on the progress of the evolution, there may be statements that have neither been processed nor are eligible to be processed.

Definition 7 (Pending set) *The* pending *set of an instance is the set of statements whose commands have been sent to their agents and whose agents have not yet responded.*

We denote the pending set by W. Then $W \subset S$.

Definition 8 (Responded set) *The* responded *set of an instance is the set of statements whose agents have responded.*

We denote the responded set by R. Then $R \subset S$.

Definition 9 (Eligible set) *If D is an instance, $\Omega = (S, \Delta)$ is its associated statement net, W is its pending set, and R is its responded set, then its* eligible *set is*

$$E = S - \Big(W \cup R \cup \bigcup_{s:\, \tilde{R}} s^{\bullet} \Big). \tag{11}$$

When instance D is created, it contains an initial process D. The context determines whether we are referring to instance D or process D. If there is any doubt we explicitly refer to instance D or process D. Process D represents the initial state of instance D. Process D evolves through a sequence of subsequent processes. $D_{(W,R,c)}$ denotes a process in the sequence, as explained in the following paragraphs. A parenthesized subscript always denotes a process, never an instance.

Instance D's state may be $free$, in which case it may receive a *start* control and begin to execute its statement net. If instance D is free, then it contains the process D. If instance D is executing its statement net, then its state is determined by its pending set W, its responded set R, and whether it is stopping. Instance D then contains a process which we represent as $D_{(W,R,c)}$, where $c = 0$ if the instance is not stopping, and 1 if it is. We call c the *stopping indicator*.

Let a_i, $i \in I$ be a channel through which agent P_i can receive commands and controls, and send the result of the last command it received. Hereafter, the term *channel* means a channel through which an instance or an agent receives commands and controls and returns responses. Agents may receive

commands and controls from an instance, another agent, or a user. For example, an instance may send a command to a coffee brewing agent, the brewing agent may request the grinding agent to provide coffee, and the user may request the brewing agent to pour previously brewed coffee.

7 Definitions

7.1 Instance definitions

Instance D initially contains the process D, and in this state it can receive a *start* control on channel b. The *start* control causes the process D to evolve to $D_{(\emptyset,\emptyset,0)}$, in which state it can execute a statement net $\Omega = (S, \Delta)$ by sending commands to designated agents and receiving their responses, evolving through a sequence of subsequent processes as it does so. We will show later that it must ultimately return to the process D.

$$D \stackrel{def}{=} b(x) \cdot [x = \underline{start}]\tau \cdot D_{(\emptyset,\emptyset,0)} \tag{12}$$

If D is executing its statement net, and it is not stopping, then it may send a command from any eligible statement to the agent designated by the statement; or receive a response from any agent associated with a statement in the pending set; or receive a *stop* control on channel b. If it receives *stop* on channel b, or it receives *fail* from any agent, it ceases processing its statement net, and sends *stop* to all pending agents. When a statement net has completed processing, that is, its state is $(\emptyset, S, c)$, the associated instance sends a response on channel b and makes a transition to its initial state. We

express this by Equations 13 to 16. As a notational aside, note that we omit the set constructors { and } that would normally enclose S_{ijl} when it occurs in subexpressions such as $R \cup S_{ijl}$ and $W - S_{ijl}$. Recall from Section 6.1 that by notational definition $S_{ijl} = ((P_i, C_j), l)$.

$$\begin{aligned} D_{(W,R,0)} \stackrel{def}{=} & \sum_{S_{ijl}\colon E} \bar{a}_i C_j \cdot D_{(W \cup S_{ijl},R,0)} + \\ & \sum_{S_{ijl}\colon W} a_i(x) \cdot [x = \underline{ok}]\tau \cdot D_{(W - S_{ijl},R \cup S_{ijl},0)} + \\ & \sum_{S_{ijl}\colon W} a_i(y) \cdot [y = \underline{fail}]\tau \cdot D_{(W - S_{ijl},R \cup S_{ijl},1)} + \\ & b(z) \cdot [z = \underline{stop}]\tau \cdot D_{(W,R,1)} \end{aligned} \tag{13}$$

$$\begin{aligned} D_{(W,R,1)} \stackrel{def}{=} & \sum_{S_{ijl}\colon W} \bar{a}_i \underline{stop} \cdot D_{(W - S_{ijl},R,1)} + \\ & \sum_{S_{ijl}\colon W} a_i(x) \cdot [x = \underline{ok}]\tau \cdot D_{(W - S_{ijl},R \cup S_{ijl},1)} + \\ & \sum_{S_{ijl}\colon W} a_i(y) \cdot [y = \underline{fail}]\tau \cdot D_{(W - S_{ijl},R \cup S_{ijl},1)} \end{aligned} \tag{14}$$

$$D_{(\emptyset,S,0)} \stackrel{def}{=} \bar{b}\underline{ok} \cdot D \tag{15}$$

$$D_{(\emptyset,R,1)} \stackrel{def}{=} \bar{b}\underline{fail} \cdot D \tag{16}$$

An instance may attempt to execute a statement containing a command referencing a busy agent. If so, no transition occurs, and the sets W, R and E are unchanged. The statement remains in E and the instance attempts to execute it later, unless it receives $\underline{stop}$ in the meantime. Our model does not

specify how an instance should attempt further execution. For example, an instance may poll the agent until it becomes free, or it may register a desire to be notified when the agent is free. This is an implementation issue.

Normally an instance executes every statement in S and evolves to $D_{(\emptyset,S,0)}$, and thence to D. However, if an instance receives *stop* or *fail*, then it evolves, ultimately, to D without evolving through $D_{(\emptyset,S,0)}$, and without necessarily evolving through $D_{(\emptyset,S,1)}$. That is, an instance receiving *stop* or *fail*, although it always returns to D, generally does not execute every statement in its statement net. This is expressed by Equation 16, which shows that $D_{(\emptyset,R,1)}$ evolves to D, although the responded set is not equal to S. Later, we prove a termination theorem (Theorem 2), which states that an instance that evolves from its initial process D to $D_{(\emptyset,\emptyset,0)}$, by receiving a *start* control, continues to evolve to the process D.

When one or more actions **may** occur, π-calculus does not require that any one of them **must** occur. However, our model requires that whenever $D_{(W,R,c)}$ **may** evolve to one or more target processes, then it **must** evolve to one of those processes. This ensures that an instance must ultimately complete the execution of its statement net, or receive *stop* or *fail*, then send a response on channel b. This does not apply to the process D; that is, there is no requirement for a *start* control to be sent to D. This is expressed by the following constraint.

Constraint 2 (Instances must evolve) *If any action described by Equations 13, 14, 15 or 16 may occur, then one of those actions must occur.*

7.2 Agent definitions

When agent P_i receives a command (on channel a_i), it executes the command and, in doing so, it may request services from other agents by executing an internal statement net Ω; if it does not require services from other agents, its statement net is $\Omega = (\emptyset, \emptyset)$. Its statement net must not contain a command addressed to itself. That is

Constraint 3 (Agent cannot send command to itself) *If $\Omega = (S, \Delta)$ is the statement net of agent P_i, then for all $j \in J$ and $l \in L$, there is no S_{ijl} in S.*

When agent P_i has completed its statement net and its own work it sends $\underline{ok}$ or $\underline{fail}$ on channel a_i. In executing these actions, agent P_i evolves through a sequence of states. The following paragraphs define its evolution.

Agent P_i's state may be $free$, in which case it may receive a command on channel a_i and begin to execute it. To execute the command, agent P_i may obtain the services of other agents by executing an internal statement net. If agent P_i is free, then it contains the process P_i. If agent P_i is executing a command, then its state is determined by its pending set W, its responded set R, and whether it is stopping. Agent P_i then contains a process which we represent as $P_{i_{(W,R,c)}}$, where $c = 0$ if the agent is not stopping, and 1 if it is. If agent P_i is executing a command that does not require services of other agents, then it contains the process $P_{(\emptyset,\emptyset,0)}$. Agent P_i receiving a command on channel a_i is represented as follows:

$$P_i \stackrel{def}{=} a_i(x) \cdot P_{i_{(\emptyset,\emptyset,0)}} \tag{17}$$

where $i \in I$, $x \notin \{\underline{start}, \underline{stop}\}$.

If agent P_i receives a command, it silently executes the command and an internal statement net (which may be $(\emptyset, \emptyset)$). When it completes both, it responds $\underline{ok}$ or $\underline{fail}$ on channel a_i and becomes free.

The following agent definitions correspond to the instance definitions in Section 7.1.

$$\begin{aligned} P_{i_{(W,R,0)}} \stackrel{def}{=} & \sum_{S_{ijl}\colon E} \bar{a}_i C_j \cdot P_{i_{(W \cup S_{ijl}, R, 0)}} + \\ & \sum_{S_{ijl}\colon W} a_i(x) \cdot [x = \underline{ok}]\tau \cdot P_{i_{(W - S_{ijl}, R \cup S_{ijl}, 0)}} + \\ & \sum_{S_{ijl}\colon W} a_i(y) \cdot [y = \underline{fail}]\tau \cdot P_{i_{(W - S_{ijl}, R \cup S_{ijl}, 1)}} + \\ & a_i(z) \cdot [z = \underline{stop}]\tau \cdot P_{i_{(W,R,1)}} \end{aligned} \tag{18}$$

$$\begin{aligned} P_{i_{(W,R,1)}} \stackrel{def}{=} & \sum_{S_{ijl}\colon W} \bar{a}_i \underline{stop} \cdot P_{i_{(W - S_{ijl}, R, 1)}} + \\ & \sum_{S_{ijl}\colon W} a_i(x) \cdot [x = \underline{ok}]\tau \cdot P_{i_{(W - S_{ijl}, R \cup S_{ijl}, 1)}} + \\ & \sum_{S_{ijl}\colon W} a_i(y) \cdot [y = \underline{fail}]\tau \cdot P_{i_{(W - S_{ijl}, R \cup S_{ijl}, 1)}} \end{aligned} \tag{19}$$

$$P_{i_{(\emptyset,S,0)}} \stackrel{def}{=} \bar{a}_i \underline{ok} \cdot P_i \tag{20}$$

$$P_{i_{(\emptyset,R,1)}} \stackrel{def}{=} \bar{a}_i \underline{fail} \cdot P_i \tag{21}$$

Note that if P_i received $\underline{stop}$ or $\underline{fail}$, then it may not have evolved to the process $P_{i_{(\emptyset,S,0)}}$.

Our model requires that whenever $P_{i_{(W,R,c)}}$ **may** evolve to one or more target processes, then it **must** evolve to one of those processes. This ensures that each agent must ultimately complete its internal work (which must end with either success or failure), or it must receive $\underline{stop}$ or $\underline{fail}$, then it must respond $\underline{ok}$ or $\underline{fail}$ on channel a_i. This does not apply to the process P_i; that is, there is no requirement for a command to be sent to P_i. This is expressed by the following constraint.

Constraint 4 (Agents must evolve) *If any action described by Equations 18, 19, 20 or 21 may occur, then one of those actions must occur.*

7.3 System definitions

We define the system consisting of an instance and agents as follows:

$$\underline{System} \stackrel{def}{=} D \Big| \prod_{i:\, I} P_i \tag{22}$$

8 Transitions

8.1 Instance transitions

$\underline{System}$ evolves as transitions occur. The following transitions describe the evolution of an instance. When process D receives $\underline{start}$, on channel b it evolves to $D_{(\emptyset,\emptyset,0)}$, whence its instance's statement net can be executed. Transitions 1 to 10 include every transition that can occur in the subsystem described by Equations 12 to 16. This can be verified by enumerating

the actions described by Equations 12 to 16 and ensuring that there is a transition corresponding to each action.

Transition 1 $$D \xrightarrow{\underline{bstart}} D_{(\emptyset,\emptyset,0)}$$

Process $D_{(W,R,0)}$ may perform any of the transitions expressed by Equation 13, evolving as shown by the following transitions.

Transition 2 $$\textit{Given } S_{ijl} \in E,\ D_{(W,R,0)} \xrightarrow{\bar{a}_i C_j} D_{(W \cup S_{ijl},R,0)}$$

Transition 3

$$\textit{Given } S_{ijl} \in W,\ D_{(W,R,0)} \xrightarrow{\underline{a_i ok}} \tau \cdot D_{(W-S_{ijl},R \cup S_{ijl},0)} \xrightarrow{\tau} D_{(W-S_{ijl},R \cup S_{ijl},0)}$$

Transition 4

$$\textit{Given } S_{ijl} \in W,\ D_{(W,R,0)} \xrightarrow{\underline{a_i fail}} \tau \cdot D_{(W-S_{ijl},R \cup S_{ijl},1)} \xrightarrow{\tau} D_{(W-S_{ijl},R \cup S_{ijl},1)}$$

Transition 5 $$D_{(W,R,0)} \xrightarrow{\underline{bstop}} \tau \cdot D_{(W,R,1)} \xrightarrow{\tau} D_{(W,R,1)}$$

Process $D_{(W,R,1)}$ may perform any of the transitions expressed by Equation 14, evolving as shown by the following transitions.

Transition 6 $$\textit{Given } S_{ijl} \in W,\ D_{(W,R,1)} \xrightarrow{\bar{a}_i \underline{stop}} D_{(W-S_{ijl},R,1)}$$

Transition 7

$$\textit{Given } S_{ijl} \in W,\ D_{(W,R,1)} \xrightarrow{\underline{a_i ok}} \tau \cdot D_{(W-S_{ijl},R \cup S_{ijl},1)} \xrightarrow{\tau} D_{(W-S_{ijl},R \cup S_{ijl},1)}$$

Transition 8

$$\textit{Given } S_{ijl} \in W,\ D_{(W,R,1)} \xrightarrow{\underline{a_i fail}} \tau \cdot D_{(W-S_{ijl},R \cup S_{ijl},1)} \xrightarrow{\tau} D_{(W-S_{ijl},R \cup S_{ijl},1)}$$

Process $D_{(\emptyset,S,0)}$ or process $D_{(\emptyset,R,1)}$ may perform any of the transitions expressed by Equation 15, or by Equation 16, evolving as shown by the following transitions.

Transition 9 $D_{(\emptyset,S,0)} \xrightarrow{\bar{b}\underline{ok}} D$

Transition 10 $D_{(\emptyset,R,1)} \xrightarrow{\bar{b}\underline{fail}} D$

8.2 Process set

Definition 10 (Process set) *The* process set *of a statement net* $\Omega = (S, \Delta)$ *is* $\{D_{(w,r,c)} | w \subseteq S \wedge r \subseteq S \wedge (c = 0 \vee c = 1)\}$.

We denote the process set by Q. Note that $D \notin Q$. Q is the set of processes that can be constructed by allowing w and r to range over all subsets of S, and the stopping indicator c to take the values 0 and 1. Q may contain processes that cannot occur; for example, from Lemma 1, which we will later prove, it can be deduced that the process $D_{(S,S,0)}$ cannot occur.

Definition 11 (Transition relation) *The* transition relation Ψ *is a relation on the process set* Q *such that* $(q_i, q_j) \in \Psi$ *if and only if there is a transition from* q_i *to* q_j.

We call the graph (Q, Ψ) the transition graph of Ω.

Definition 12 (Reachable process set) *The* reachable process set Q' *is the subset of processes of* Q *reachable by a path from* $D_{(\emptyset,\emptyset,0)}$.

8.3 Agent transitions

The following transitions describe the evolution of agents. Transitions 11 to 20 include every transition that can occur in the subsystem described by Equations 17 to 21. This can be verified by enumerating the actions described by Equations 17 to 21 and ensuring that there is a transition corresponding to each action.

When process P_i receives any input (which can only be a command), on channel a_i it evolves to a process in which its agent's statement net can be executed, as follows

Transition 11 $$P_i \xrightarrow{a_i x} P_{i_{(\emptyset,\emptyset,0)}}$$

Process $P_{i_{(W,R,0)}}$ may perform any of the transitions expressed by Equation 18, evolving as shown by the following transitions.

Transition 12 $$\textit{Given } S_{ijl} \in E,\ P_{i_{(W,R,0)}} \xrightarrow{\bar{a}_i C_j} P_{i_{(W \cup S_{ijl},R,0)}}$$

Transition 13

$$\textit{Given } S_{ijl} \in W,\ P_{i_{(W,R,0)}} \xrightarrow{a_i \underline{ok}} \tau \cdot P_{i_{(W-S_{ijl},R \cup S_{ijl},0)}} \xrightarrow{\tau} P_{i_{(W-S_{ijl},R \cup S_{ijl},0)}}$$

Transition 14

$$\textit{Given } S_{ijl} \in W,\ P_{i_{(W,R,0)}} \xrightarrow{a_i \underline{fail}} \tau \cdot P_{i_{(W-S_{ijl},R \cup S_{ijl},1)}} \xrightarrow{\tau} P_{i_{(W-S_{ijl},R \cup S_{ijl},1)}}$$

Transition 15 $$P_{i_{(W,R,0)}} \xrightarrow{\underline{bstop}} \tau \cdot P_{i_{(W,R,1)}} \xrightarrow{\tau} P_{i_{(W,R,1)}}$$

Process $P_{i_{(W,R,1)}}$ may perform any of the transitions expressed by Equation 19, evolving as shown by the following transitions.

Transition 16 $\quad Given\ S_{ijl} \in W,\ P_{i_{(W,R,1)}} \xrightarrow{\bar{a}_i \underline{stop}} P_{i_{(W-S_{ijl},R,1)}}$

Transition 17

$$Given\ S_{ijl} \in W,\ P_{i_{(W,R,1)}} \xrightarrow{a_i \underline{ok}} \tau \cdot P_{i_{(W-S_{ijl},R\cup S_{ijl},1)}} \xrightarrow{\tau} P_{i_{(W-S_{ijl},R\cup S_{ijl},0)}}$$

Transition 18

$$Given\ S_{ijl} \in W,\ P_{i_{(W,R,1)}} \xrightarrow{a_i \underline{fail}} \tau \cdot P_{i_{(W-S_{ijl},R\cup S_{ijl},1)}} \xrightarrow{\tau} P_{i_{(W-S_{ijl},R\cup S_{ijl},1)}}$$

Process $P_{i_{(\emptyset,S,0)}}$ or process $P_{i_{(\emptyset,R,1)}}$ may perform any of the transitions expressed by Equation 20, or by Equation 21, evolving as shown by the following transitions.

Transition 19 $\quad P_{i_{(\emptyset,S,0)}} \xrightarrow{\bar{a}_i \underline{ok}} P_i$

Transition 20 $\quad P_{i_{(\emptyset,R,1)}} \xrightarrow{\bar{a}_i \underline{fail}} P_i$

Table 1 shows a possible evolution of the system used in the running example introduced in Section 4. Other evolutions are also possible. (The silent transitions τ have been omitted.)

8.4 Statement net termination

Since every statement net is finite and by Constraints 2 and 4, either an instance/agent receives *stop*, or every command ultimately ends in success or failure, the process $D_{(W,R,0)}$ ultimately evolves to either $D_{(\emptyset,S,0)}$ or $D_{(\emptyset,R,1)}$, both of which, by Equation 15 or 16, evolve to D. Similarly for agents, the process $P_{i_{(W,R,0)}}$ ultimately evolves to either $P_{i_{(\emptyset,S,0)}}$ or $P_{i_{(\emptyset,R,1)}}$, which, by

Table 1: Example: a possible evolution

$$D \xrightarrow{\underline{bstart}} D_{(\emptyset,\emptyset,0)}$$

$$D_{(\emptyset,\emptyset,0)} \xrightarrow{\bar{a}_1(\underline{make},\underline{nullArg})} D_{(\{S_{111}\},\emptyset,0)}$$

$$P_1 \xrightarrow{a_1(\underline{make},\underline{nullArg})} P1_{(\emptyset,\emptyset,0)}$$

$$P1_{(\emptyset,\emptyset,0)} \xrightarrow{\bar{a}_1\underline{ok}} P_1$$

$$D_{(\{S_{111}\},\emptyset,0)} \xrightarrow{a_1\underline{ok}} D_{(\emptyset,\{S_{111}\},0)}$$

$$D_{(\emptyset,\{S_{111}\},0)} \xrightarrow{\bar{a}_2(\underline{turn},\underline{dim})} D_{(\{S_{222}\},\{S_{111}\},0)}$$

$$P_2 \xrightarrow{a_2(\underline{turn},\underline{dim})} P2_{(\emptyset,\emptyset,0)}$$

$$P2_{(\emptyset,\emptyset,0)} \xrightarrow{\bar{a}_2\underline{ok}} P_2$$

$$D_{(\{S_{222}\},\{S_{111}\},0)} \xrightarrow{a_2\underline{ok}} D_{(\emptyset,\{S_{111},S_{222}\},0)}$$

$$D_{(\emptyset,\{S_{111},S_{222}\},0)} \xrightarrow{\bar{a}_3(\underline{show},\underline{news})} D_{(\{S_{333}\},\{S_{111},S_{222}\},0)}$$

$$P_3 \xrightarrow{a_3(\underline{show},\underline{news})} P3_{(\emptyset,\emptyset,0)}$$

$$P3_{(\emptyset,\emptyset,0)} \xrightarrow{\bar{a}_3\underline{ok}} P_3$$

$$D_{(\{S_{333}\},\{S_{111},S_{222}\},0)} \xrightarrow{a_3\underline{ok}} D_{(\emptyset,\{S_{111},S_{222},S_{333}\},0)}$$

$$D_{(\emptyset,\{S_{111},S_{222},S_{333}\},0)} \xrightarrow{\bar{a}_2(\underline{turn},\underline{bright})} D_{(\{S_{244}\},\{S_{111},S_{222},S_{333}\},0)}$$

$$P_2 \xrightarrow{a_2(\underline{turn},\underline{bright})} P2_{(\emptyset,\emptyset,0)}$$

$$D_{(\{S_{244}\},\{S_{111},S_{222},S_{333}\},0)} \xrightarrow{\bar{a}_1(\underline{make},\underline{nullArg})} D_{(\{S_{244},S_{115}\},\{S_{111},S_{222},S_{333}\},0)}$$

$$P_1 \xrightarrow{a_1(\underline{make},\underline{nullArg})} P1_{(\emptyset,\emptyset,0)}$$

$$P2_{(\emptyset,\emptyset,0)} \xrightarrow{\bar{a}_2\underline{ok}} P_2$$

$$D_{(\{S_{244},S_{115}\},\{S_{111},S_{222},S_{333}\},0)} \xrightarrow{a_2\underline{ok}} D_{\{(S_{115}\},\{S_{111},S_{222},S_{333},S_{244}\},0)}$$

$$P1_{(\emptyset,\emptyset,0)} \xrightarrow{\bar{a}_1\underline{ok}} P_1$$

$$D_{\{(S_{115}\},\{S_{111},S_{222},S_{333},S_{244}\},0)} \xrightarrow{a_1\underline{ok}} D_{(\emptyset,\{S_{111},S_{222},S_{333},S_{244},S_{115}\},0)}$$

$$D_{(\emptyset,\{S_{111},S_{222},S_{333},S_{244},S_{115}\},0)} \xrightarrow{\bar{b}\underline{ok}} D$$

Equation 20 or 21, evolve to P_i. We formalize this with Theorem 2 at the end of this Section. The theorem is stated in terms of an instance, but a similar theorem can be proved for agents. Before stating the theorem, we introduce the following lemmas, which will be useful in proving the theorem. In the following, D is an instance; $\Omega = (S, \Delta)$ is its associated statement net; at some point in the evolution of Ω's processes, W is its pending set, R is its responded set, E is its eligible set, and c indicates whether it is stopping; Q' is the reachable process set. To prove several of the lemmas, we inspect every defined transition to verify whether or not it can occur under the conditions that hold in the statement of the lemma.

We form the subsets of Q' defined in Table 2. For example, Q_5 is the set of processes $D_{(W,R,c)}$ such that $W = \emptyset$, $\emptyset \subset R \subset S$, $c = 1$ and $D_{(W,R,c)}$.

The pending set and the responded set are disjoint.

Lemma 1 $W \cap R = \emptyset$

Proof: This follows from Definitions 7 and 8. ■

Lemma 2 $Q_9 = \emptyset$ *and* $Q_{10} = \emptyset$.

Proof: By Lemma 1, W and R are disjoint. Since $R = S$, there cannot be a member of S in W. ■

If there is at least one statement that has not responded, and no statement is pending, then there is at least one eligible statement. This is expressed by the following Lemma.

Lemma 3 $(R \subset S$ *and* $W = \emptyset) \implies E \neq \emptyset$.

Table 2: A partition of Q'

Q_1	$W = \emptyset$	$R = \emptyset$	c $= 0$	$D_{(W,R,c)} \in Q'$
Q_2	$W = \emptyset$	$R = \emptyset$	c $= 1$	$D_{(W,R,c)} \in Q'$
Q_3	$W = \emptyset$	$R = S$	c $= 0$	$D_{(W,R,c)} \in Q'$
Q_4	$W = \emptyset$	$R = S$	c $= 1$	$D_{(W,R,c)} \in Q'$
Q_5	$W = \emptyset$	$\emptyset \subset R \subset S$	c $= 0$	$D_{(W,R,c)} \in Q'$
Q_6	$W = \emptyset$	$\emptyset \subset R \subset S$	c $= 1$	$D_{(W,R,c)} \in Q'$
Q_7	$W \neq \emptyset$	$R = \emptyset$	c $= 0$	$D_{(W,R,c)} \in Q'$
Q_8	$W \neq \emptyset$	$R = \emptyset$	c $= 1$	$D_{(W,R,c)} \in Q'$
Q_9	$W \neq \emptyset$	$R = S$	c $= 0$	$D_{(W,R,c)} \in Q'$
Q_{10}	$W \neq \emptyset$	$R = S$	c $= 1$	$D_{(W,R,c)} \in Q'$
Q_{11}	$W \neq \emptyset$	$\emptyset \subset R \subset S$	c $= 0$	$D_{(W,R,c)} \in Q'$
Q_{12}	$W \neq \emptyset$	$\emptyset \subset R \subset S$	c $= 1$	$D_{(W,R,c)} \in Q'$

Table 3: Simple cycles of Γ

A	Q_7, Q_7	$W+$
B	Q_{12}, Q_{12}	$W-$
C	Q_{11}, Q_{11}	$W+$
D	Q_{11}, Q_{11}	$W-, R+$
E	Q_8, Q_8	$W-$
F	Q_{11}, Q_5, Q_{11}	$R+$

Proof: Assume E is empty. Then, from Definition 9, $S = s_1^\bullet \cup s_2^\bullet \cup \cdots \cup s_n^\bullet$ where $\tilde{R} = \{s_1, s_2, \ldots, s_n\}$. Hence, each member of S occurs in at least one of $s_i^\bullet, i = 1, 2, \ldots, n$. From Definition 5, $s_i^\bullet = \{x\colon S | (s_i, x) \in \Delta\}, i = 1, 2, \ldots, n$. Therefore, for each member $y \in S$ there is a member $y' \in \tilde{R}$ such that $(x', x) \in \Delta$. There are two cases: (a) $R \neq \emptyset$ and (b) $R = \emptyset$.

Consider case (a). Choose an arbitrary member $y \in R$. Since $R \subset S$, it follows that there is a member $y' \in \tilde{R}$ such that $(y', y) \in \Delta$. Since y has responded, its associated command has been sent to an agent, and therefore no prerequisite path on which it lies contains a non-responded statement that precedes y. However, (y', y) is a component of a prerequisite path, and y' precedes y, and y' has not responded, since it is in $\tilde{R}$. It follows that, for case (a), E is not empty.

Now, consider case (b), that is $R = \emptyset$. Choose an arbitrary member $y \in S$. Then there is a member $y' \in \tilde{R}$ such that $(y', y) \in \Delta$. Since $\tilde{R} = S$, $y' \in S$. Hence, for all $y \in S$ there is a $y' \in S$ such that $(y', y) \in \Delta$. Therefore Ω is not acyclic. It follows that, for case (b), E is not empty. ■

If the eligible set is not empty and the system is not stopping, then transition 2 can occur and, if it does, then the cardinality of the pending set increases, the responded set is unchanged, and the system is not stopping.

Lemma 4 *If $E \neq \emptyset$ and $c = 0$, then transition 2 can occur and, if it does, $card(W)$ increases, R is unchanged, and $c = 0$.*

Proof: This follows from Transition 2. ■

If the pending set is not empty, and the system is not stopping, then tran-

sition 3, or 4 can occur and, if either transition does, then the cardinality of the pending set decreases and the cardinality of the responded set increases.

Lemma 5 *If $W \neq \emptyset$ and $c = 0$, then transition 3 can occur and, if it does, $card(W)$ decreases, $card(R)$ increases and $c = 0$.*

Proof: This follows from Transition 3. ■

Lemma 6 *If $W \neq \emptyset$ and $c = 0$, then transition 4 can occur and, if it does, $card(W)$ decreases, $card(R)$ increases, and $c = 1$.*

Proof: This follows from Transition 4. ■

If instance D is not stopping, then it may receive *stop*, which moves the system to a state in which it is stopping; the pending set and responded set are unchanged.

Lemma 7 *If $c = 0$ then transition 5 can occur and, if it does, W is unchanged, R is unchanged, and $c = 1$.*

Proof: This follows from Transition 5. ■

Lemma 8 *If $W \neq \emptyset$ and $c = 1$, then transition 6 can occur and if it does, $card(W)$ decreases, R is unchanged, and $c = 1$.*

Proof: This follows from Transition 6. ■

Lemma 9 *If $W \neq \emptyset$ and $c = 1$, then transition 7 or 8 can occur and, if either transition does occur, $card(W)$ decreases, $card(R)$ increases, and $c = 1$.*

Proof: This follows from Transitions 7 and 8. ■

Lemma 10 *Transition 1 cannot occur from any process in any of* $Q_1, \ldots, Q_{12}$.

Proof: Transition 1 cannot occur because the initial process is not D. ■

Lemma 11 *If* $E = \emptyset$, *then transition 2 cannot occur.*

Proof: This follows from Transition 2. ■

Lemma 12 *If* $c = 1$, *then transitions 2, 3, 4 and 5 cannot occur.*

Proof: This follows from Transitions 2, 3, 4 and 5. ■

Lemma 13 *If* $c = 0$, *then transitions 6, 7 and 8 cannot occur.*

Proof: This follows from Transitions 6, 7 and 8. ■

Lemma 14 *If* $W = \emptyset$, *then transitions 3, 4, 6, 7 and 8 cannot occur.*

Proof: Transitions 3, 4, 6, 7 and 8 cannot occur because there is no given $S_{ijl} \in W$. ■

Lemma 15 *Transitions 9 and 10 do not transit to any process in any of* $Q_1, \ldots, Q_{12}$.

Proof: This follows from Transitions 9 and 10. ■

Lemma 16 *The subsets* $Q_1, \ldots, Q_{12}$ *defined in Table 2 are a partition of* Q'.

Proof: We need to show that $Q_1, \ldots, Q_{12}$ *are pair-wise disjoint, and their union is* Q'.

An inspection of each subset reveals that no member of Q' *occurs in more than one subset. Hence* $Q_1, \ldots, Q_{12}$ *are pair-wise disjoint.*

Table 2 partitions the domain of values from which W *is drawn into two subsets: empty and non-empty; it partitions the domain of values from which*

R is drawn into three subsets: empty, S and neither; it partitions the domain of values from which c is drawn into two subsets: 0 and 1. If the triple (W, R, c) *is formed by drawing one value from each domain, then there are 12 distinct ways in which the triple can be formed. All 12 ways are represented in Table 2.* Q' *is the union of* $Q_1, \ldots, Q_{12}$. ■

As a step in proving the termination theorem, we define the labelled graph Γ as follows. Let Φ be a ternary relation such that $(Q_i, Q_j, t) \in \Phi$ if and only if t is a transition from some process in Q_i to some process in Q_j. Let Γ be a labelled graph representing Φ, where the nodes are members of $\{Q_1, \ldots, Q_{12}\}$ and each edge (Q_i, Q_j) is labelled with t. Let each edge of Γ be labelled with 0, 1 or 2 additional labels as follows: $W-$ if and only if t decreases $card(W)$; $W+$ if and only if t increases $card(W)$; and $R+$ if and only if t increases $card(R)$. We now prove that Γ is the labelled graph represented in Figure 2.

Lemma 17 *The labelled graph represented in Figure 2 is* Γ.

Proof: We verify that Figure 2 contains an edge for each transition that can occur, and no other edge; and that the edges are labelled as described.

Consider Q_1. *Initially* $W = \emptyset$ *and* $R = \emptyset$ *and* $c = 0$. *By Lemma 7, transition 5 can occur and, if it does,* $W = \emptyset$, $R = \emptyset$ *and* $c = 1$. *Hence the labelled graph contains* (Q_1, Q_2), *which has none of the additional labels* $W+$, $W-$ *or* $R+$. *By Lemma 3,* $E \neq \emptyset$; *hence, by Lemma 4, transition 2 can occur and, if it does,* $W \neq \emptyset$, $R = \emptyset$ *and* $c = 0$. *Hence the labelled graph contains* (Q_1, Q_7), *which has the additional label* $W+$ *only. By Lemma 10, transition 1 cannot occur. By Lemma 14, transitions 3, 4, 6, 7 and 8 cannot occur.*

By Lemma 15, transitions 9 and 10 do not transit to any process in any of $Q_1, \ldots, Q_{12}$*. Hence the labelled graph contains no other edges emanating from* Q_1*.*

Consider Q_2*. Initially* $W = \emptyset$ *and* $R = \emptyset$ *and* $c = 1$*. By Lemma 10, transition 1 cannot occur. By Lemma 12, transitions 2, 3, 4 and 5 cannot occur. By Lemma 14, transitions 6, 7 and 8 cannot occur. By Lemma 15, transitions 9 and 10 do not transit to any process in any of* $Q_1, \ldots, Q_{12}$*. Hence the labelled graph contains no edges emanating from* Q_2*.*

Consider Q_3*. Initially* $W = \emptyset$ *and* $R = S$ *and* $c = 0$*. By Lemma 7, transition 5 can occur and, if it does,* $W = \emptyset$*,* $R = S$ *and* $c = 1$*. Hence the labelled graph contains* (Q_3, Q_4)*, which has none of the additional labels* $W+$*,* $W-$ *or* $R+$*. By Lemma 10, transition 1 cannot occur. By Definition 9,* $E = \emptyset$*; hence, by Lemma 11, transition 2 cannot occur. By Lemma 14, transitions 3, 4, 6, 7 and 8 cannot occur. By Lemma 15, transitions 9 and 10 do not transit to any process in any of* $Q_1, \ldots, Q_{12}$*. Hence the labelled graph contains no other edges emanating from* Q_3*.*

Consider Q_4*. Initially* $W = \emptyset$ *and* $R = S$ *and* $c = 1$*. By Lemma 10, transition 1 cannot occur. By Lemma 12, transitions 2, 3, 4 and 5 cannot occur. By Lemma 14, transitions 6, 7 and 8 cannot occur. By Lemma 15, transitions 9 and 10 do not transit to any process in any of* $Q_1, \ldots, Q_{12}$*. Hence the labelled graph contains no edges emanating from* Q_4*.*

Consider Q_5*. Initially* $W = \emptyset$ *and* $\emptyset \subset R \subset S$ *and* $c = 0$*. By Lemma 7, transition 5 can occur and, if it does,* $W = \emptyset$*,* $\emptyset \subset R \subset S$ *and* $c = 1$*. Hence the labelled graph contains* (Q_5, Q_6)*, which has none of the additional labels*

$W+$, $W-$ or $R+$. By Lemma 3, $E \neq \emptyset$; hence, by Lemma 4, transition 2 can occur and, if it does, $W \neq \emptyset$, $\emptyset \subset R \subset S$ and $c = 0$. Hence the labelled graph contains (Q_5, Q_{11}), which has the additional label $W+$. By Lemma 10, transition 1 cannot occur. By Lemma 14, transitions 3, 4, 6, 7 and 8 cannot occur. By Lemma 15, transitions 9 and 10 do not transit to any process in any of $Q_1, \ldots, Q_{12}$. Hence the labelled graph contains no other edges emanating from Q_5.

Consider Q_6. Initially $W = \emptyset$ and $\emptyset \subset R \subset S$ and $c = 1$. By Lemma 10, transition 1 cannot occur. By Lemma 12, transitions 2, 3, 4 and 5 cannot occur. By Lemma 14, transitions 6, 7 and 8 cannot occur. By Lemma 15, transitions 9 and 10 do not transit to any process in any of $Q_1, \ldots, Q_{12}$. Hence the labelled graph contains no edges emanating from Q_6.

Consider Q_7. Initially, $W \neq \emptyset$ and $R = \emptyset$ and $c = 0$. If $E \neq \emptyset$, by Lemma 4, transition 2 can occur, and, if it does, $W \neq \emptyset$, $card(W)$ increases, $R = \emptyset$, and $c = 0$. Hence, the labelled graph contains (Q_7, Q_7), which has the additional label $W+$. Whether E is empty or not, by Lemma 5, transition 3 can occur, and, if it does, $card(W)$ decreases, $card(R)$ increases, and $c = 0$. Hence the labelled graph contains (Q_7, Q_3) (if finally $W = \emptyset$ and $R = S$), (Q_7, Q_5) (if finally $W = \emptyset$ and $R \subset S$), and (Q_7, Q_{11}) (if finally $W \neq \emptyset$ and $R \subset S$), all of which have the additional labels $W-$ and $R+$. Whether E is empty or not, by Lemma 6, transition 4 can occur, and, if it does, $card(W)$ decreases, $card(R)$ increases, and $c = 1$. Hence the labelled graph contains (Q_7, Q_4) (if finally $W = \emptyset$ and $R = S$), (Q_7, Q_6) (if finally $W = \emptyset$ and $R \subset S$), and (Q_7, Q_{12}) (if finally $W \neq \emptyset$ and $R \subset S$), all of which have

the additional labels $W-$ and $R+$. Whether E is empty or not, by Lemma 7, transition 5 can occur, and, if it does, W is unchanged, R is unchanged and $c = 1$. Hence the labelled graph contains (Q_7, Q_8), which has none of the additional labels $W+$, $W-$ or $R+$. By Lemma 10, transition 1 cannot occur. By Lemma 13, transitions 6, 7 and 8 cannot occur. By Lemma 15, transitions 9 and 10 do not transit to any process in any of $Q_1, \ldots, Q_{12}$. Hence the labelled graph contains no other edges emanating from Q_7.

Consider Q_8. Initially, $W \neq \emptyset$ and $R = \emptyset$ and $c = 1$. By Lemma 8, transition 6 can occur, and, if it does, $card(W)$ decreases, $R = \emptyset$, and $c = 1$. Hence the labelled graph contains (Q_8, Q_2) (if finally $W = \emptyset$) and (Q_8, Q_8) (if finally $W \neq \emptyset$), both of which have the additional label $W-$. By Lemma 9, transitions 7 and 8 can occur, and, if either does, $card(W)$ decreases, $card(R)$ increases, $\emptyset \subset R$, and $c = 1$. Hence the labelled graph contains (Q_8, Q_4) (if finally $W = \emptyset$ and $R = S$), (Q_8, Q_6) (if finally $W = \emptyset$ and $R \subset S$), and (Q_8, Q_{12}) (if finally $W \neq \emptyset$ and $R \subset S$), all of which have the additional labels $W-$ and $R+$. By Lemma 10, transition 1 cannot occur. By Lemma 12, transitions 2, 3, 4, 5, cannot occur. By Lemma 15, transitions 9 and 10 do not transit to any process in any of $Q_1, \ldots, Q_{12}$. Hence the labelled graph contains no other edges emanating from Q_8.

Consider Q_9. Here, $W \neq \emptyset$ and $R = S$ and $c = 0$. By Lemma 2, Q_9 is empty. Hence the labelled graph contains no edges entering Q_9 or emanating from Q_9.

Consider Q_{10}. Here, $W \neq \emptyset$ and $R = S$ and $c = 1$. By Lemma 2, Q_{10} is empty. Hence the labelled graph contains no edges entering Q_{10} or emanating

from Q_{10}.

Consider Q_{11}. *Initially,* $W \neq \emptyset$ *and* $\emptyset \subset R \subset S$ *and* $c = 0$. *If* $E \neq \emptyset$, *by Lemma 4, transition 2 can occur, and, if it does,* $W \neq \emptyset$, $card(W)$ *increases,* $R \subset S$, *and* $c = 0$. *Hence, the labelled graph contains* (Q_{11}, Q_{11}), *which has the additional label* $W+$. *Whether* E *is empty or not, by Lemma 5, transition 3 can occur, and, if it does,* $card(W)$ *decreases,* $card(R)$ *increases, and* $c = 0$. *Hence the graph contains* (Q_{11}, Q_3) *(if finally* $W = \emptyset$ *and* $R = S$*),* (Q_{11}, Q_5) *(if finally* $W = \emptyset$ *and* $R \subset S$*), and* (Q_{11}, Q_{11}) *(if finally* $W \neq \emptyset$ *and* $R \subset S$*), all of which have the additional labels* $W-$ *and* $R+$. *Whether* E *is empty or not, by Lemma 6, transition 4 can occur, and, if it does,* $card(W)$ *decreases,* $card(R)$ *increases, and* $c = 1$. *Hence the labelled graph contains* (Q_{11}, Q_4) *(if finally* $W = \emptyset$ *and* $R = S$*),* (Q_{11}, Q_6) *(if finally* $W = \emptyset$ *and* $R \subset S$*), and* (Q_{11}, Q_{12}) *(if finally* $W \neq \emptyset$ *and* $R \subset S$*), all of which have the additional labels* $W-$ *and* $R+$. *Whether* E *is empty or not, by Lemma 7, transition 5 can occur, and, if it does,* W *is unchanged,* R *is unchanged and* $c = 1$. *Hence the labelled graph contains* (Q_{11}, Q_{12}), *which has none of the additional labels* $W+$, $W-$ *or* $R+$. *By Lemma 10, transition 1 cannot occur. By Lemma 13, transitions 6, 7 and 8 cannot occur. By Lemma 15, transitions 9 and 10 do not transit to any process in any of* $Q_1, \ldots, Q_{12}$. *Hence the labelled graph contains no other edges emanating from* Q_{11}.

Consider Q_{12}. *Initially,* $W \neq \emptyset$ *and* $\emptyset \subset R \subset S$ *and* $c = 1$. *By Lemma 8, transition 6 can occur and, if it does,* $card(W)$ *decreases,* $\emptyset \subset R \subset S$, *and* $c = 1$. *Hence the labelled graph contains* (Q_{12}, Q_6) *(if finally* $W = \emptyset$*) and* (Q_{12}, Q_{12}) *(if finally* $W \neq \emptyset$*). By Lemma 9, transition 7 or 8 can occur and,*

if either does, card(W) decreases, card(R) increases, and c = 1. Hence, the labelled graph contains (Q_{12}, Q_4) *(if finally* $W = \emptyset$ *and* $R = S$*),* (Q_{12}, Q_6) *(if finally* $W = \emptyset$ *and* $R \subset S$*) and* (Q_{12}, Q_{12}) *(if finally* $W \neq \emptyset$ *and* $R \subset S$*). By Lemma 10, transition 1 cannot occur. By Lemma 12, transitions 2, 3, 4 and 5 cannot occur. By Lemma 15, transitions 9 and 10 do not transit to any process in any of* $Q_1, \ldots, Q_{12}$*. Hence the labelled graph contains no other edges emanating from* Q_{12}*.* ■

We are now in a position to prove the following theorem.

Theorem 2 *If instance D's initial process D evolves to* $D_{(\emptyset,\emptyset,0)}$ *(by receiving a* start *control), then it will continue to evolve to the process D.*

Proof: Suppose D has evolved to $D_{(\emptyset,\emptyset,0)}$*. Form the subsets* $Q_1, \ldots, Q_{12}$ *of* Q' *shown in Table 2. By Lemma 16,* $Q_1, \ldots, Q_{12}$ *are a partition of* Q'*.* $D_{(\emptyset,\emptyset,0)} \in Q_1$*. Consider the labelled graph* Γ *defined above. By Lemma 17, Figure 2 represents* Γ*. If instance D has evolved to some process in* $Q_1, Q_5, Q_7, Q_8, Q_{11}, Q_{12}$*, then, by Constraint 2, it must continue to evolve. Its continued evolution must take it to a process in one of* $Q_1, Q_5, Q_7, Q_8, Q_{11}, Q_{12}$ *or in one of* Q_2, Q_3, Q_4, Q_6*. If it continues indefinitely to evolve to processes only in* $Q_1, Q_5, Q_7, Q_8, Q_{11}, Q_{12}$*, then it must traverse one or more of the simple cycles of* Γ*. The simple cycles are shown in Table 3, which indicates the effect on card(R) and card(W) when the cycle is traversed. Each simple cycle contains transitions that either increase card(R), or increase card(W), or decrease card(W). card(R) has an upper bound (namely card(S)) and cannot increase indefinitely. card(W) has both lower and upper bounds (*0

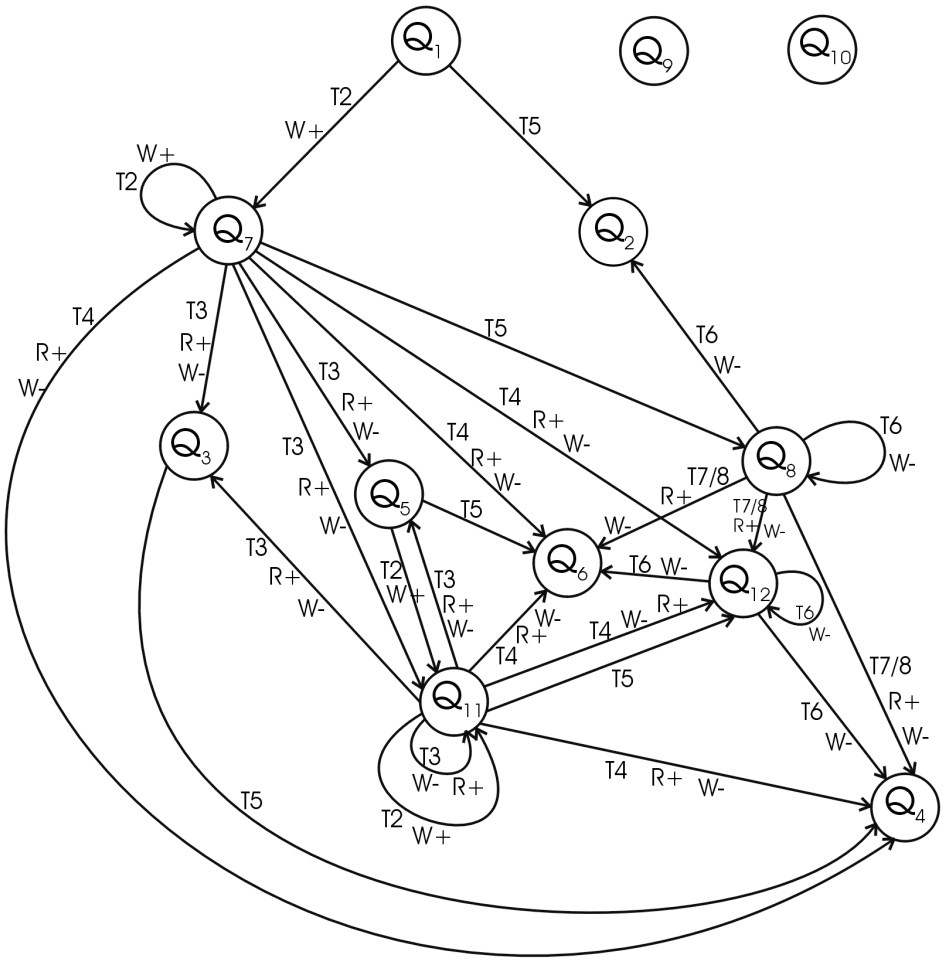

Figure 2: The labelled graph Γ

and $card(S)$). $card(W)$ can neither increase indefinitely nor decrease indefinitely, but it may experience a series of increases intermingled with a series of decreases. We now show that any cycle containing intermingled $card(W)$ increases and decreases must also contain at least one $card(R)$ increase, and therefore cannot be traversed indefinitely. The simple cycles that increase $card(W)$ are A and C. The simple cycles that decrease $card(W)$ are B, D and E. (Note that cycle F leaves $card(W)$ unchanged, but increases $card(R)$.) Consider the simple cycles that increase $card(W)$. No cycle can contain both the simple cycle A and any cycle that decreases $card(W)$, because, after leaving Q_7, there is no path returning to Q_7. Hence, any cycle that contains intermingled $card(W)$ increases and decreases must contain the simple cycle C. The only cycles that contain both C and a simple cycle that decreases $card(W)$ also contain simple cycle D. This cycle increases $card(R)$ and cannot be traversed indefinitely. Hence, instance D must ultimately evolve to a process in one of Q_2, Q_3, Q_4, Q_6. From Q_3 it can evolve to either process D or to a process in Q_4 and, from Constraint 2, it must evolve to one of these. A process in one of Q_2, Q_4 or Q_6 can only evolve to process D and, from Constraint 2, it must evolve to D. ■

A *deadlock* occurs if a statement net, as defined in Section 7.3, evolves to a process in which no further transition is possible. A *livelock* occurs if a statement net evolves to a process from which there is no evolution to D.

Corollary 1 *A statement net contains no deadlocks or livelocks.*

Proof: Theorem 2 shows that it always evolves to D. ■

8.5 Instance and agent comparisons

Instances and agents have simliarities and differences, as described below.

Given statement net. Instances and agents are both associated with a given statement net.

start **control.** An instances executes its statement net when it receives a *start* control. An agent executes its statement net when it receives a command.

Control source. An instance receives a *start* control from outside the System (on channel b). An agent receives a command from inside the System (from an instance or another agent).

Internal functionality. An instance has no internal functionality; it executes its statement net by sending commands and controls to designated agents. An agent has internal functionality; it executes its received command by internally performing designated actions; to perform an action, it executes its internal statement net to send commands to other agents.

User invocation. An instance is invoked by a user only to start or stop execution of its statement net. An agent may be invoked directly by a user with any valid command.

9 Conclusions and future work

Our model has provided a foundation for what it means to have devices (e.g., appliances) working together in an orchestrated way, formalizing the metaphor of workflow applied to device ecologies. On such a foundation, eco languages can be designed as a high-level abstraction for programming device ecologies, with appropriate termination properties. We believe that our model could provide a basis on which standards could ultimately be specified for communication among devices. Such standards would enable manufacturers to incorporate features in their products that would add value to the users' experience when disparate devices are interconnected in a suitable environment.

There are several areas where our model can be extended. The set *response* need not be limited to the members $\underline{ok}$ and $\underline{fail}$. We have constructed an example in which other responses would be useful. Further, there is little difference between an instance and an agent. If we regard an instance as a special case of an agent (one that has a non-empty statement net and no other internal functionality) then some simplification could be introduced by combining the instance and agent equations and removing the distinction between them. Instances could then communicate with each other by treating other instances as agents.

Fault handling could be introduced by defining actions that occur in the presence of errors. For example, agents could be grouped in such a way that, for certain commands, any agent in the group could be substituted for a

faulty agent.

We intend next to develop software to model an ecology by animating potential message flows and transitions. As part of this work, we will compute an upper bound on the number of messages that will be exchanged between agents and instances for a given statement net.

References

[1] Association of Home Appliance Manufacturers. *Connected Home Appliances Object Modelling, CHA-1-2002*, 2002. Available at http://www.aham.org/.

[2] J. Bentham. *TCP/IP Lean: Web Servers for Embedded Systems (2nd Edition)*. CMP Books, 2002.

[3] H. Bohn, A. Bobek, and F. Golatowski. SIRENA - Service Infrastructure for Real time Embedded Networked Applications:a Service-Oriented Framework for Different Domains. In *Proceedings of the ITEA 2006 Conference*, June 2004.

[4] M.H. Butler. Using Capability Profiles for Appliance Aggregation. Technical Report HPL-2002-173, HP Labs, June 2002.

[5] Y. Durand, S.P.J.-M. Vincent, C. Marchand, F.-G. Ottogalli, V. Olive, S. Martin, B. Dumant, and S. Chambon. SIDRAH: A Software Infras-

tructure for a Resilient Community of Wireless Devices. In *Proceedings of the Smart Objects Conference (SOC'03)*, Grenoble, May 2003.

[6] U. Glasser, Y. Gurevich, and M. Veanes. High-Level Executable Specification of the Universal Plug and Play Architecture. In *Proceedings of the 35th Annual Hawaii International Conference on System Sciences (HICSS'02)-Volume 9*, 2002.

[7] F. Jammes, A. Mensch, and H. Smit. Service-Oriented Device Communications Using the Device Profiles for Web Services. In *Proceedings of the 3rd International Workshop on Middleware for Pervasive and Ad-Hoc Computing (MPAC05)*, Nov-Dec 2005.

[8] N. Kohtake, K. Matsumiya, K. Takashio, and H. Tokuda. Smart Device Collaboration for Ubiquitous Computing Environment. In *Proceedings of the Workshop on Multi-Device Interface for Ubiquitous Peripheral Interaction at the 5th International Conference on Ubiquitous Computing (UbiComp'03)*, Oct 2003.

[9] R. Kumar, V. Poladian, I. Greenberg, A. Messer, and D. Milojicic. Selecting Devices for Aggregation. In *Proceedings of the WMCSA 2003 (to appear)*, 2003.

[10] S.W. Loke. Service-Oriented Device Ecology Workflows. In M. Orlowska, S. Weerawarana, M.P. Papazoglou, and J. Yang, editors, *Proceedings of the International Conference on Service-Oriented Computing, Lecture*

Notes in Computer Science 2910, pages 559–574, Trento, Italy, December 2003. Springer-Verlag.

[11] R. Masuoka, B. Parsia, and Y. Labrou. Task Computing - the Semantic Web meets Pervasive Computing. In *Proceedings of the 2nd International Semantic Web Conference (ISWC 2003)*, Florida, USA, October 2003.

[12] K. Matsuura, T. Haraa, A. Watanabe, and T. Nakajima. A New Architecture for Home Computing. In *Proceedings of the IEEE Workshop on Software Technologies for Future Embedded Systems (WSTFES03)*, pages 71–74, May 2003.

[13] Sun Microsystems. *Jini Network Technology*, 2001. Available at http://wwws.sun.com/software/jini/.

[14] Robin Milner. *Communication and mobile systems: the π-calculus.* Cambridge University Press, 1999.

[15] M.W. Newman, J.Z. Sedivy, W.K. Edwards, T. Smith, K. Marcelo, C.M. Neuwirth, J.I. Hong, and S. Izadi. Designing for Serendipity: Supporting End-User Configuration of Ubiquitous Computing Environments. In *Proceedings of the Conference on Designing Interactive Systems (DIS2002)*, 2002. Available at http://www.cs.
berkeley.edu/~jasonh/publications/dis2002-speakeasy
-browser.pdf.

[16] O. Omojokun and P. Dewan. A High-Level and Flexible Framework for Dynamically Composing Networked Devices. In *Proceedings of the 5th IEEE Workshop on Mobile Computing Systems and Applications (WMCSA 2003)*, 2003.

[17] Mrio Rodrigues, Antnio Teixeira, and Luis Seabra Lopes. An Hybrid Approach for Spoken Natural Language Understanding Applied to a Mobile Intelligent Robot. In Bernadette Sharp, editor, *Proceedings of the 1st International Workshop on Natural Language Understanding and Cognitive Science*, Portugal, April 2004.

[18] Davide Sangiorgi and David Walker. *The π-calculus: a theory of mobile processes*. Cambridge University Press, 2001.

[19] J.P. Sousa and D. Garlan. From Computers Everywhere to Tasks Anywhere: The Aura Approach. In *Submitted*, 2003. Available at http://www-2.cs.cmu.edu/
˜aura/docdir/sg01.pdf.

[20] UPnP Forum. *UPnP Device Architecture*, June 2000. Available at http://www.upnp.org/.

[21] E. Vildjiounaite, E. Malm, J. Kaartinen, and P. Alahuhta. Networking of Smart Things in a Smart Home. In *Proceedings of the Workshop on the Interaction of HCI and Systems Issues in UbiComp (UBIHCISYS 2003) at the 5th International Conference on Ubiquitous Computing (UbiComp'03)*, Oct 2003. Available at http://ubihcisys.stanford.edu/online-

proceedings/
Ubi03w7-Vildjiounaite-final.pdf.