Functional Principles of Registry-based Service Discovery

V.Sundramoorthy[1], C.Tan[2], P.H.Hartel[1], J.I.den Hartog[1], and J.Scholten[1]

[1]University of Twente, Enschede, The Netherlands
vasughi.sundramoorthy@utwente.nl
[2]Massachusetts Institute of Technology (MIT), Cambridge, USA
c.tan@mit.edu

Abstract

As Service Discovery Protocols (SDP) are becoming increasingly important for ubiquitous computing, they must behave according to predefined principles. We present the functional Principles of Service Discovery for robust, registry-based service discovery. A methodology to guarantee adherence to these principles is provided and illustrated by formal verification of the principles against FRODO, an SDP built for the home environment. We show that, to make behavioral guarantees, an SDP has to be robust against network disturbances, and cannot rely only on the network layer.

1. Introduction

In the pursuit of a ubiquitous world, network administration, as we know it will change. Self-configuring systems will replace user configuration, where devices discover their environment, detect and adapt to topology changes, establish communication with each other and share services. Several state of the art systems have been developed towards achieving this objective, including Jini [1] by Sun Microsystems, Universal Plug and Play (UPnP) [9] by Microsoft and Service Location Protocol [2, 7] by the IETF.

In general, service discovery protocols adhere to certain fundamental attributes. To start with, there are two types of entities in the system: *User* and *Manager*. A Manager is a service provider, which has a set of services. Each service is represented as a *Service Description (SD)*, which describes the service in terms of: (1) device type (e.g. printer), (2) service type (e.g. color printing) and (3) attribute list (e.g. location, paper size). A User is an entity that has a set of requirements for the services it needs.

There are two types of service discovery architectures: *registry-based* (e.g. Jini) and *peer-to-peer* (e.g. UPnP). A registry-based architecture has a third entity, called the *Registry*. A Manager *registers* its services at a Registry, and Users discover the services through unicast *queries* to the Registry. In the peer-to-peer architecture there are no Registries, and Users discover Managers through broadcast or multicast queries. The registry-based architecture reduces network traffic and makes a network more manageable by allowing Registries to keep track of arriving and departing services. The peer-to-peer architecture avoids single point of failure problems, as may exist in the registry-based architecture, but increases network traffic. A combination of these two architectures can be implemented to allow the protocol to be more resilient against failure on the registry, and reduce network traffic (e.g. SLP, FRODO). However, unlike existing protocols, FRODO [10] implements resource-awareness. The service discovery tasks are partitioned according to resource constraints, where a resource-lean node depends on more powerful neighbors to complement its discovery tasks. From this point onwards, we focus on Registry-based protocols because the peer-to-peer architecture is a simplification of the registry-based architecture, and can be derived by removing Registry-related factors from this work.

Service discovery protocols perform garbage collection of defunct services. A popular mechanism for garbage collection is *leasing*. The Manager refreshes a "lease" periodically to notify the Registries of its con-

Proceedings of the IEEE Conference on Local Computer Networks 30th Anniversary (LCN'05)
0-7695-2421-4/05 $20.00 © 2005 IEEE

tinued existence. The services of Managers that have left the network can be purged automatically if the lease is not renewed.

To ensure subscribed Users remain consistent with a Manager whenever its service changes, most service discovery protocols implement update functionalities. The change can be reflected directly in the SD, or in an evented variable that gives the status of the service.

Contribution: The contributions of this paper to the field of service discovery are (1) stating for the first time, the functional principles for achieving robust registry-based service discovery, (2) classification of the functions in service discovery, (3) presenting a methodology for verifying service discovery protocols and (4) illustrating the above using our resource-aware and robust FRODO protocol.

Section 2 classifies service discovery mechanisms into four major functions, and the principles that correspond to each of the functions. Section 3 provides a brief description of FRODO. Section 4 describes the modeling and verification methodology of FRODO using DT-Spin, and Section 5 analyzes the results of verification. The last section concludes.

2. The Principles of Service Discovery

Service discovery systems that aim to be self-configuring should provide reliability in the face of network disturbances. This basically means that the service discovery functions are able to recover from errors. A system is flawed if a User is never able to discover a Manager in the system whose services match the User's requirements. Thus, we feel it is imperative for the field of service discovery to have a set of principles that makes the notion of correct behavior precise. We call these *Service Discovery Principles*. Besides ensuring reliability, these fundamental principles define the nature and constraints of service discovery. Systems that adhere to the principles will provide guarantees on their behaviors.

2.1. Related Work

The only other work done in this area is the *Service Guarantees* from NIST in [4], which proposes general guarantees that service discovery protocols should strive to satisfy. Our Service Discovery Principles refine these guarantees and moreover, we provide a methodology to proof using model-checking techniques, that a protocol satisfies these principles.

2.2. Service Discovery Functional Areas

Logically, service discovery is divided into 4 *functional areas*:

1) Configuration Discovery - Allows registry-based systems to (a) auto-configure a Registry and (b) discover the Registry. Auto-configuration of Registries requires nodes to appoint Registries, according to some criteria, on the fly, without user interventions. For Registry discovery, nodes can *actively* initiate the discovery or listen *lazily* to Registry announcements.

2) Service Registration - Allows Managers to register their services at a Registry. Registration mechanisms include (a) unsolicited registration, where nodes request the Registry to register their services and (b) solicited registration, where Registries detect and request new nodes to register.

3) Service Discovery - Allows Users to discover Managers that satisfy their set of requirements. This could be achieved through a (a) unicast query to a Registry, (b) broadcast query, (c) notification of new services by the Registry to the Users or through broadcast announcements by Managers.

4) Configuration Update - After a User discovers a service, it can subscribe to the Manager to receive updates when the service changes, or becomes unavailable. The Managers provide two types of update information, service description update and evented variable update. The updates can be propagated using (a) 2-party scheme, where the communication is directly between the Manager and the User, or (b) 3-party scheme, where a registry propagates the update to the User on behalf of the Manager. Garbage collection of defunct services is also part of the update functionality.

Different service discovery systems implement different mechanisms to achieve the objectives of the functional areas, as depicted in Table 1.

2.3. The Service Discovery Environment

During the operations of service discovery tasks, the system could experience various network disturbances, such as link failure, interface failure, message loss or node failure. Network disturbances cause uncertainties in service discovery systems. However, it is important to ensure that a service discovery system guarantees recovery from a disturbance.

Below we provide a formal description of a system.
System: A system consists of a set of entities with attributes $(e \in)\mathsf{E}$, a set of services $(s \in)\mathsf{S}$ and time in

Proceedings of the IEEE Conference on Local Computer Networks 30th Anniversary (LCN'05)
0-7695-2421-4/05 $20.00 © 2005 IEEE

Functional Area, related Principles and supportive attributes	Mechanisms	SLP	Jini	FRODO
Configuration Discovery (*Registry Setup and Registry Discovery Principles*)	Auto-configured Registry	No	No	Yes - through leader election among 300D nodes
	Registry Discovery	Active and lazy discovery	Active and lazy discovery	Active and lazy discovery
Registration (*Registration Principle*)	Registration mechanism	Unsolicited registration	Unsolicited registration	Solicited and unsolicited registration
Service Discovery (*Service Discovery Principle*)	Search mechanism	Unicast (multicast possible)	Unicast	Unicast (multicast possible)
	Notification of new services to Users	No	Yes - unicast notification by Registry	Yes - unicast notification by Registry
Configuration update (*2-party and 3-party Configuration Update Principles*)	Garbage collection	Managers renew lease	Managers renew lease	300D Managers renew lease, Registry polls 3D/3C Managers
	Update information	Service update	Service update	Service update, event notification
	Update mechanism	None - register again	3-party subscription	2-party subscription, 3-party subscription
Supportive attributes	Acknowledgements and retransmissions	TCP-dependent	TCP dependent	Implemented with timers
	Resource awareness	No	No	Yes, device classification
	Service usage	Application layer	Mobile proxy code	Application layer
	Security	Limited, with application driven authetication	None, depends on JDK 1.2 security model	None, future work

Table 1. Taxonomy of registry-based service discovery systems. FRODO classifies devices into 3C, 3D and 300D, in order of increasing resources.

progress ($t \in$)Time. The attributes of the entities, described below, evolve over time. Based on the attributes, the entities are divided into three (not necessarily pairswise disjoint) sets ($u \in$)U for Users, ($m \in$)M for Managers, and ($c \in$)C for Registries.

Entity attributes: Each entity, e has the following attributes, which are subject to change over time:

- $\mathsf{C}(e) \subseteq \mathsf{E}$ is the set of Registries discovered by e. An entity is called a Registry, i.e. $e \in \mathsf{C}$, if it has discovered itself in the role of Registry, $e \in \mathsf{C}(e)$.
- $\mathsf{OfferedSD}(e) \subseteq \mathsf{S}$ is the set of services, offered by e. An entity is called a Manager, i.e. $e \in \mathsf{M}$, if it offers at least one service, $\mathsf{OfferedSD}(e) \neq \emptyset$.
- $\mathsf{Requirement}(e) \subseteq \mathsf{S}$ is the set of services required by entity e. An entity is called a User, i.e. $e \in \mathsf{U}$, if it is interested in/requires at least one service, $\mathsf{Requirement}(e) \neq \emptyset$.
- $\mathsf{DiscoveredSD}(e, e') \subseteq \mathsf{S}$ is the set of services discovered by e at entity e'. This attribute is (typically) only used for e, a User and e', a Manager. We put $\mathsf{DiscoveredSD}(u) := \bigcup \mathsf{DiscoveredSD}(u, m)$ for the set of all services discovered by a User, u at any Manager, m.
- $\mathsf{RegisteredSD}(e) \subseteq \mathsf{S}$ is the set of registered services in e. This attribute is only used for a Registry.

There are several protocol-dependent parameters used in the Service Discovery Principles:

Connectivity condition: $\mathsf{Conn}(e, e')$: The service discovery protocol is responsible for providing the definition of Connectivity. An example is "if a message is transmitted from either e to e', or vice versa , the message is received." The Connectivity condition is not restricted to valid communication paths. It can also be defined by an application, for example, a security application that detects a malicious entity, and indicates to the service discovery protocol that the node is not available for any further operations.

Disconnect condition: $\mathsf{DisConn}(e, e') : \neg\mathsf{Conn}(e, e')$ for "sufficiently long", where "sufficiently long" is a

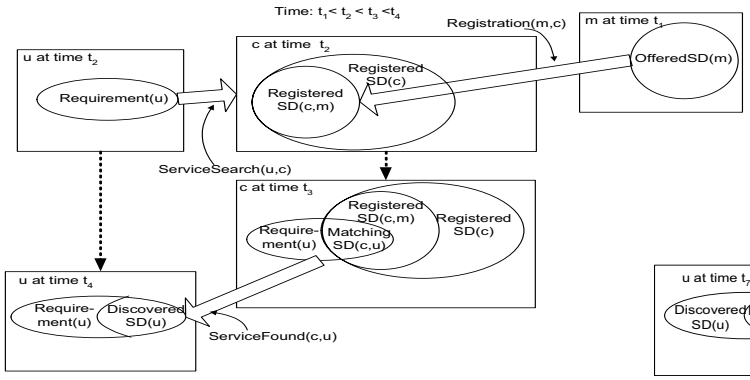

Figure 1. System flow and relations between sets during Registration and Service Discovery. *Registration, ServiceSearch* **and** *ServiceFound* **are shown as messages sent between entities. A service is registered by the Manager at time t_1, then discovered by the Registry at t_2, and User searches for the service at or before t_2. The Registry processes the request at t_3, where it finds matching services for the User. The User discovers the service at t_4.**

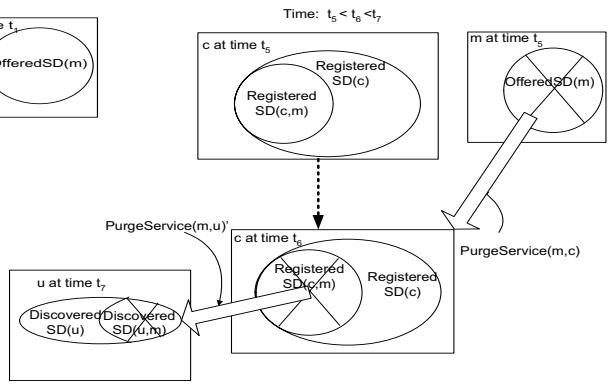

Figure 2. System flow and relations between sets during Configuration Purge. A service is purged by the Manager at t_5, then the Registry is notified at t_6, which then purges the registration. The Registry notifies the User at t_7, and the User purges the service from its DiscoveredSD **cache.**

protocol-dependent parameter. The definition of "sufficiently long" includes the Connectivity definition and the period communication did not occur. Examples are the limit of retransmissions or the time period for waiting for acknowledgements.

Global Connectivity, GC: The condition is satisfied if all entities are connected.

$\mathsf{GC} : \forall \mathsf{e}_1, \mathsf{e}_2 \in \mathsf{E} : \mathsf{Conn}(\mathsf{e}_1, \mathsf{e}_2)$

Number of Registries, N: the desired number of Registries in the system.

Required Registries, $\mathsf{G(e)}(\subseteq \mathsf{E})$: the Registries required by e. An entity may not be interested in knowing all Registries but only those of a certain type, i.e. those which satisfy its "Registry requirements".

Registry election, Rank: a function for electing the Registries in the system, where the set C has highestRank if $\neg(\exists \mathsf{e}' \notin \mathsf{C} : \mathsf{Rank}(\mathsf{e}') > \min\{\mathsf{Rank}(\mathsf{e}) \mid \mathsf{e} \in \mathsf{C}\})$

2.4. Service Discovery Principles

We use *Linear Temporal Logic (LTL)* [8] to define the Service Discovery Principles. This is essentially positional logic with temporal operators such as $\Box$ (*always*) or $\Diamond$ (*eventually*). Basically the recovery of a service discovery system is defined as the *response* pattern, $\Box(p' \rightarrow \Diamond p)$, where p' is the state of Connectivity, Global Connectivity or Disconnect, and p is the response of the service discovery system towards satisfying a Service Discovery Principle.

We need some auxiliary definitions before giving the Service Discovery Principles.

a) The symbols "$\Diamond\subseteq$", "$\Diamond\supseteq$" and "$\Diamond\supset\subset$"
$\mathsf{a} \Diamond\subseteq \mathsf{b}$ holds at time t_0 when a at time t_0 is a subset of b at some time t_1 where $\mathsf{t}_1 \geq \mathsf{t}_0$. Similiarly, for the symbols "$\Diamond\supseteq$" and "$\Diamond\supset\subset$" where $\mathsf{a} \supset\subset \mathsf{b}$ denotes that the two sets are disjoint ($\mathsf{a} \cap \mathsf{b} = \emptyset$).

b) $\mathsf{ServiceSearch(u, c)}$ states that a User, u is looking for a specific service from a Registry, c

c) $\mathsf{MatchingSD(c, u)} :=$
$\mathsf{RegisteredSD(c)} \cap \mathsf{Requirement(u)}$
is the set of services registered at the Registry, c which match the requirements of User, u.

d) $\mathsf{ServiceFound(c, u)} :=$
$\mathsf{MatchingSD(c, u)} \Diamond\subseteq \mathsf{DiscoveredSD(u)}$
states that the matching services at a Registry c are discovered by the User u.

e) $\mathsf{Registration(m, c)} :=$
$\mathsf{OfferedSD(m)} \Diamond\subseteq \mathsf{RegisteredSD(c)}$
states that all services of Manager, m are registered at Registry, c.

f) $\mathsf{PurgeService(m, u)} :=$
$\mathsf{OfferedSD(m)} \Diamond\supset\subset \mathsf{DiscoveredSD(u)}$

Proceedings of the IEEE Conference on Local Computer Networks 30th Anniversary (LCN'05)
0-7695-2421-4/05 $20.00 © 2005 IEEE

states that the services of Manager, m are eventually purged from the discovered services of User, u.
$\mathsf{PurgeService}(\mathsf{m},\mathsf{c}) :=$
$\mathsf{OfferedSD}(\mathsf{m}) \lozenge\supset\subset \mathsf{RegisteredSD}(\mathsf{c})$
similarly states that the services of Manager m are eventually purged from the registered services of Registry, c. If an entity is both a Registry and a User then both properties must hold.

g) $\mathsf{Uptodate}(\mathsf{u},\mathsf{m}) := \mathsf{OfferedSD}(\mathsf{m}) \cap \mathsf{Requirement}(\mathsf{u}) \lozenge\subseteq \mathsf{DiscoveredSD}(\mathsf{u},\mathsf{m}) \wedge \mathsf{OfferedSD}(\mathsf{m}) \lozenge\supseteq \mathsf{DiscoveredSD}(\mathsf{u},\mathsf{m})$
states that User u will find new services at Manager m and remove services that are no longer available at m.

Figures 1 and 2 provide scenarios of a registry-based system, where the relationship between entities, and the sets are explained, as time progresses. In Figure 1, ServiceSearch is the message providing the requirements of the User to the Registry, while in Figure 2, PurgeService and PurgeService′ are the messages that notify the Registry and User respectively of a defunct service.

With the system and basic properties in place we are ready to give the 7 *Service Discovery Principles*.

(P1) Registry Setup Principle
When there is global connectivity, N Registries of the highest rank are selected in the system.
$\Box(\mathsf{GC} \rightarrow \lozenge((|\mathsf{C}| = \mathsf{N}) \wedge \mathsf{highestRank}))$
Assume all entities have the same rank if no ranking function is used.

(P2) Registry Discovery Principle
An entity discovers all available Registries in the system that it is interested in.
$\forall \mathsf{e} : \Box(\mathsf{C}(\mathsf{e}) \subseteq \mathsf{G}(\mathsf{e}) \wedge \forall \mathsf{c} \in \mathsf{C} \cap \mathsf{G}(\mathsf{e}) : \mathsf{Conn}(\mathsf{c},\mathsf{e}) \rightarrow \lozenge \mathsf{c} \in \mathsf{C}(\mathsf{e}))$

(P3) Registration Principle
A Manager registers its service description at each Registry it discovers.
$\forall \mathsf{m}, \forall \mathsf{c} \in \mathsf{C}(\mathsf{m}) : \Box(\mathsf{Conn}(\mathsf{m},\mathsf{c}) \rightarrow \mathsf{Registration}(\mathsf{m},\mathsf{c}))$

(P4) Registry-based Service Discovery Principle
A User discovers the services that match the User's requirements and which are registered at a Registry.
$\forall \mathsf{u}, \forall \mathsf{c} \in \mathsf{C}(\mathsf{u}) : \Box(\mathsf{Conn}(\mathsf{u},\mathsf{c}) \rightarrow \mathsf{ServiceFound}(\mathsf{c},\mathsf{u}))$

(P5) Configuration Purge Principle
A User or Registry purges the services of a Manager that has become disconnected.
$\forall \mathsf{m}, \forall \mathsf{uc} \in \mathsf{U} \cup \mathsf{C} : \Box(\mathsf{DisConn}(\mathsf{m},\mathsf{uc}) \rightarrow \mathsf{PurgeService}(\mathsf{m},\mathsf{uc}))$

(P6) 2-Party Configuration Update Principle:
A User remains consistent with a Manager when its services change.
$\forall \mathsf{m}, \mathsf{u} : \Box(\mathsf{Conn}(\mathsf{u},\mathsf{m}) \rightarrow \mathsf{Uptodate}(\mathsf{u},\mathsf{m}))$

(P7) 3-Party Configuration Update Principle
A User remains consistent with a Manager when its services change, through the Registry.
$\forall \mathsf{c} \in \mathsf{C}(\mathsf{m}) : \Box((\mathsf{Conn}(\mathsf{c},\mathsf{m}) \wedge \lozenge \mathsf{Conn}(\mathsf{u},\mathsf{c})) \rightarrow \mathsf{Uptodate}(\mathsf{u},\mathsf{m}))$

The 7 Service Discovery Principles state the fundamental objectives of service discovery; the User discovers an available Manager, which matches its service requirement, and achieve consistency with the Manager when the service changes.

3. FRODO

FRODO implements resource-awareness and robustness. For resource-awareness, FRODO classifies nodes as (1) 3C device class - simple devices with restricted resources (e.g. smart dust), (2) 3D device class- medium complex devices (e.g. temperature controller) and (3) 300D device class - powerful devices, controlled by a complex embedded computer, with more than 1MB memory requirement (e.g. set-top boxes). Only a 300D node can become a Registry, through a *leader election* process. The Registry is called the *Central*. The functions of service discovery explained in Section 2.4 are implemented according to the device classes. The names of the functions remain the same for FRODO, except we classify Configuration Update as part of the wider Configuration Management, to accommodate its robust features. Table 1 provides a summary of the mechanisms that each function implements. The protocol is less dependent than the state-of-the-art on the recovery ability of the lower layers. This allows the protocol to be deployed together with leaner lower layer protocol stacks, with restricted error recovery mechanisms. More details on FRODO can be found in [10].

4. Modeling and Verification

We use model checking [6] to verify that FRODO adheres to the Service Discovery Principles. If there are cases where the protocol fails the verification, we identify whether the error lies in the modeling process, or in the design phase. The former requires the model to be corrected (and often our understanding!). However, it is the latter which is most beneficial, since detecting a design flaw leads towards discovering mechanisms that

improve and strengthen the protocol, until the Service Discovery Principles are satisfied.

We verify the desired behavioral properties of the protocol through exhaustive enumeration (explicit or implicit) of all the states reachable by the system and the behaviors that navigate through them. We use the model-checker DT-Spin [3], for this purpose. DT-Spin is an extension of the well-known SPIN tool [6]. DT-Spin is used instead of the original SPIN model checker because we need multiple timeouts that occur at any time, even if other processes are being executed in the model. Timeouts are essential requirements for the recovery processes of the protocol against failures. The *timeout* variable in standard SPIN cannot be used as it is only true when the system is idle. However, DT-Spin also increases state space because time is modeled as ticks, and each tick occupies a state. A reasonable number of ticks is set for each timer, to make verification feasible.

4.1. Methodology

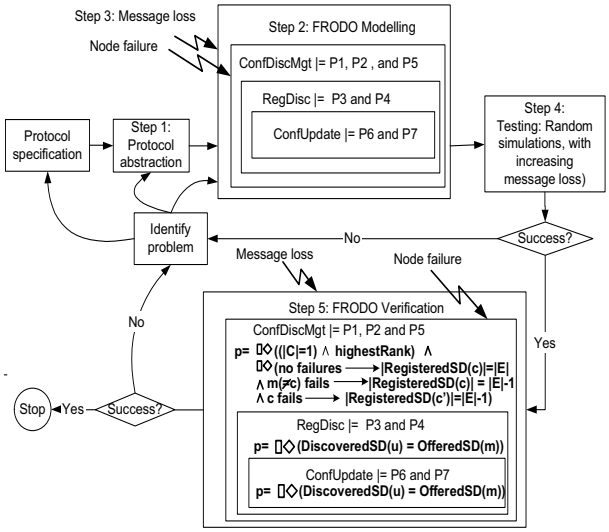

Figure 3. Methodology steps. The FRODO Modeling box shows the abstraction link between the 3 modules, where the outer boxes abstract some functions from the inner boxes. Connectivity and Global Connectivity are modeled with/without message loss respectively, and Disconnect is modeled as node failure.

It is impractical to work with monolithic models of complex protocols such as FRODO. In the first place error traces would be too cluttered with irrelevant detail to be able to spot the real problems, and secondly the state space would grow beyond the bounds of what the current tools can cope with. Therefore we split the FRODO protocol in a number of modules, each of which corresponds roughly to one of the four functional areas or sub-functions thereof. Each model is then provided in several versions, including a concrete model with the most detail, some versions that represent worst case behavior, and a number of abstract versions with as little detail as possible. By combining a concrete version of one module with appropriate abstract versions of all others, the system as a whole can be verified, focusing on the behavior of the concrete module. The main challenge of this method is to keep the different versions of each module consistent. Figure 3 shows the methodology that we use to model, simulate, and verify FRODO against the Service Discovery Principles.

Step 1: Protocol abstraction. Each assembly of modules builds on a common layer of protocol abstraction. We deliberately abstract irrelevant detail such as message format.

Step 2: Modular decomposition. a) *ConfDiscMgt* models Configuration Discovery and Configuration Management. This part is actually broken into four sub-functions as follows: *ConfDiscMgt-1* models the leader election protocol which elects one Central. *ConfDiscMgt-2* models two worst-case scenarios of the protocol with (a) all nodes claiming to be Central after a network partitioning, and (b) all registration information in the Central being lost because of prolonged communication failure. *ConfDiscMgt-3* models a Backup taking over as the Central when the existing Central leaves the system. This model uses node failure, instead of message loss to model network disturbance. *ConfDiscMgt-4* models the Central handing over to a superior node that enters late into the system. This model is an abstraction of ConfDiscMgt-1, where message loss is already checked, and therefore not included in ConfDiscMgt-4. b) *SrvRegDisc* models both Registration and Service Discovery functions. Configuration Discovery is abstracted away in this model. The model consists of one 300D node which is the Central. Service discovery is done through the Central using the directed search mechanism, thus the service cannot be discovered unless registration occurs. The discovery of each type of Manager (3C, 3D and 300D) is modeled separately. c) *ConfUpdate* models Configuration Update propagating the updates to the Users through the 2-party and 3-party subscription mechanisms.

Step 3: Message loss. Network disturbance is modeled as message loss. All models except ConfDiscMgt-3 and ConfDiscMgt-4 are simulated and verified with and without message loss. A receiver will continue executing its tasks, unaware of any message loss, which may lead towards a violation of the Service Discovery Principles. We use a counter for every message *type*, which increments whenever a particular type of message is lost. The message may be lost, until a constant *MAX_LOSS* for its type is reached. The following is an example of how a *SrvRegReq* message is sent or lost. The variables *rcvrID, OfferedSD*, and *srcID* are the receiver node's identifier, the service identifier that represents a service and the sender node's identifier respectively.

```
if
:: loss_counter[type]<= MAX_LOSS ->
          lossCounter[type]++
:: else -> /* transmit message */
     send!SrvRegReq,OfferedSD,srcID,rcvrID;
fi
```

Message loss increases the number of states because of the additional counter and timing variables and nondeterministic choice. The impact of message loss on the verification is shown in Table 2.

Step 4: Testing. We use the simulator tool in DT-Spin to test and debug the models. We test different, random simulation scenarios and increase the MAX_LOSS for every simulation (up to 10 message losses to capture extreme scenarios)

Step 5: Verification. DT-Spin checks correctness claims that are generated from logic formulas expressed in LTL. When a claim is invalid for a model, the tool produces a counter example that explicitly shows how the property was violated. The counter example is a feedback for the simulator tool of DT-Spin to trace the execution trail that causes the violation.

4.2. Property Modeling

The following are the interpretation of the protocol-dependent parameters:

1. Connectivity condition: $\mathsf{Conn}(\mathsf{e}, \mathsf{e}')$: *if a message is transmitted from either* e *to* e'*, or vice versa, and the expected acknowledgement arrives, the entities are reachable.*
2. Disconnect condition: $\neg\mathsf{Conn}(\mathsf{e}, \mathsf{e}')$: *for twice the timeout period.*
3. Rank, a leader election function that returns the highest ranked 300D node as the Central.
4. $\mathsf{G}(\mathsf{e}) = \mathsf{E}$ the nodes are interested in any Central as there is only one type of Central in FRODO.

Model	State vector (bytes)	Depth	States stored
ConfDiscMgt-1, exhaustive mode, no message loss	304	37328	38345
ConfDiscMgt-1, supertrace mode, MAX_LOSS=1	308	61993	497,662 billion

Table 2. An example of the impact of message loss on state space.

5. $\mathsf{N} = 1$, a single Central is elected.

We model the response pattern $\Box(\mathsf{p}' \rightarrow \Diamond\mathsf{p})$ defined in Section 2.4 by building the property p' directly into the models, so that p is verified as a *recurrence property* [6]. *The recurrence property* $\Box\Diamond\mathsf{p}$ *states that if the state formula,* p *happens to be false at any given point in a run, it is always guaranteed to become true again if the run is continued.* For $\mathsf{p}' =$ Global Connectivity, we verify p in a model without any message loss. For $\mathsf{p}' =$ Connectivity, we verify p in a model with a limit on message loss. For $\mathsf{p}' =$ Disconnect, we verify p in a model with node failure.

The descriptions of each property, p are given below. For the purpose of readability, we left out some technical details.

$\mathsf{ConfDiscMgt} \models \mathsf{P1} \wedge \mathsf{P2} \wedge \mathsf{P5}$
For P1, all entities have to agree on a the highest ranking node, say c, becoming the single Central. For P2, all nodes must discover this Central (and no others). In the verification we check $|\mathsf{RegisteredSD(c)}| = |\mathsf{E}|$ which implies that each node m has registered at c As each node will offer exactly one service, it will only discover one Central and will only register at this Central. For P5, the Central purges service registration of disconnected nodes. Thus, $|\mathsf{RegisteredSD(c)}| = |\mathsf{E}| - 1$. If the failing node is the Central itself then the Backup c', which is the second highest ranking node, must detect this and take over as the new Central, resulting in $|\mathsf{RegisteredSD}(\mathsf{c}')| = |\mathsf{E}| - 1$. Thus we check property

$\mathsf{p} := \Box\Diamond($ no failures $\rightarrow |\mathsf{RegisteredSD(c)}| = |\mathsf{E}| \wedge$
$\mathsf{m}(\neq \mathsf{c})$ fails $\rightarrow |\mathsf{RegisteredSD(c)}| = |\mathsf{E}| - 1 \wedge$
c fails $\rightarrow |\mathsf{RegisteredSD}(\mathsf{c}')| = |\mathsf{E}| - 1)$

Together with some basic properties of the behavior of each of the nodes (e.g. discovering only one Central) only registering at a node it believes to be the Central, ...) this is sufficient to obtain that P1, P2 and P5 all hold.

$\mathsf{SrvRegDisc} \models \mathsf{P3} \wedge \mathsf{P4}$
We consider the situation where a User u is interested in the service that is offered by some Manager m in the system, $\mathsf{Requirement(u)} = \mathsf{OfferedSD(m)}$. We verify the property
$\mathsf{p} := \Box\Diamond(\mathsf{DiscoveredSD(u)} = \mathsf{OfferedSD(m)})$
Which gives that services can be found (P4) and also that services are registered (P3) as the User can only discover registered services.

$\mathsf{ConfUpdate} \models \mathsf{P6} \wedge \mathsf{P7}$
We consider the situation where the service at m discovered by a User u changes but still satisfies the requirements of the User. To satisfy P6 and P7, the User has to update its discovered services to remain consistent with the Manager, $\mathsf{Uptodate(u, m)}$. This is implied by $\mathsf{DiscoveredSD(u)} = \mathsf{OfferedSD(m)}$. We again check the property
$\mathsf{p} := \Box\Diamond(\mathsf{DiscoveredSD(u)} = \mathsf{OfferedSD(m)})$
where $\mathsf{OfferedSD(m)}$ changes during the verification.

5. Results & Discussion

5.1. Verification results

The results that we obtained are summarized in Table 3. For every module, we provide the setting for the verification. There are multiple MAX_LOSS values. This is because for every module, there are several models, which we verified for the same properties. In total, we developed and verified 21 models, out of which, 17 models achieved Exhaustive coverage (100% coverage of all reachable states for a model), while 4 models had to be run under the Supertrace mode (uses bit state hashing [5], with coverage around 98%) because of state space explosion.

The successful results shown in Table 3 are under the assumptions that (1) the nodes provide correct information, (2) message losses are low, and (3) the system does not require a fixed time constraint on satisfying the Service Discovery Principles. Based on these assumptions, the results show that FRODO guarantees that the functions of service discovery meet their objectives under the condition that nodes are able to communicate eventually. The results of the verification increases confidence in FRODO's capabilities.

5.2. Analyzing Service Discovery Behavior

In service discovery, the system can oscillate from an ideal state, to a non-ideal state, for every service discovery function, where a network disturbance is the trigger for the transition. For example, a node can successfully discover the Registry, but fail to register when the registration message is lost. Thus the Registry Setup and Registry Discovery Principles are satisfied, but the Registration Principle is violated. The protocol must ensure that the Registration function can return to the ideal state, and satisfy the principle.

Several mechanisms are vital to satisfy the Service Discovery Principles. These are: (1) periodic lazy discovery - periodic announcements from the Registries ensure detection of other Registries, to recover from network partitioning. A service discovery system can use this scheme to either converge into N Registries, or Registries can synchronize their configuration information. (2) Re-registration - a previous successful registration does not necessarily mean it remains valid. The Registry can face node failure, or can purge the knowledge of the Manager because of network failures. There are several mechanisms that can resolve this issue: (a) periodic leasing with the Registry, a popular concept for most service discovery systems. The use of this method alone violates the Registration and Registry-based Service Discovery Principles if the Registry fails, and the Manager does not attempt to register with any other available Registries, (b) unsolicited registration, where nodes monitor Registry announcements to detect an unknown Registry, and initiate registration, and (c) solicited registration, where the Registry requests unknown nodes to register through monitoring of periodic active announcements. (3) Ensuring update propagation - acknowledgements can help in ensuring an update is propagated successfully. However, if the Manager undergoes another change before the first update is propagated, it is likely that the Registry and the User will never be aware of the first change. Polling the Manager when an expected update is missed (e.g. through message sequence) ensures that the updates are always propagated successfully.

In practical implementations, a protocol that violates some of the Service Discovery Principles can address its incompleteness through network and application layers. For example in SLP, service update is done through the application layer, where interested Users can periodically poll the Manager to receive service changes. In Jini, acknowledgements and retransmissions are handled by the TCP layer. This creates a dependency on a reliable communication channel and restricts Jini to certain types of networks. However, delegating tasks away from the service discovery layer leaves the system vulnerable to ambiguous interpretation on failure response.

Module	Settings	Principle	Results
ConfDiscMgt, without solicited registration and lazy discovery	MAX_LOSS=1 to 4, $\mathsf{N} = 1$, $\mathsf{E} = 4$ for ConfDiscMgt-1 and ConfDiscMgt-2 $\mathsf{E} = 3$ for ConfDiscMgt-3 and $\mathsf{E} = 5$ for ConfDiscMgt-4	Registry Setup, Registry Discovery, Configuration Purge	Fail
ConfDiscMgt, with solicited registration and lazy discovery	MAX_LOSS=1 to 4, $\mathsf{N} = 1$, $\mathsf{E} = 4$ for ConfDiscMgt-1 and ConfDiscMgt-2, $\mathsf{E} = 3$ for ConfDiscMgt-3 and $\mathsf{E} = 5$ for ConfDiscMgt-4,	Registry Setup, Registry Discovery, Configuration Purge	Success
SrvRegDisc	Nodes=3 (Central, Manager, User), MAX_LOSS=1, $\lvert\mathsf{OfferedSD(m)}\rvert = 1$	Registration, Service Discovery	Success
ConfUpdate	Nodes=4 (Central, Manager, User), MAX_LOSS=1, $\lvert\mathsf{OfferedSD(m)} = 1\rvert$, then 2,	Configuration Update	Success
Verify models during unconnected state	Nodes=4 300Ds, message loss not limited	Disconnect check on all principles.	Success. Principles still hold

Table 3. Verification results. Failures are corrected by implementing additional mechanisms

6. Conclusions

We classify the behavior of registry-based service discovery into 4 functional areas, and analyze the mechanisms used in each of these areas. We show that existing registry based service discovery protocols fit into this classification.

The functional requirements of service discovery, needed for correct behavior of applications in the ubiquitous environment, are expressed in 7 Service Discovery Principles. Existing protocols do not satisfy these principles because they rely on the underlying network to provide robustness. In contrast, our own service discovery protocol FRODO does satisfy the principles. We show this by formal modeling and verification.

FRODO is the first service discovery protocol that provides formally verified functional guarantees. The simulation and verification process is beneficial in discovering flaws in the design and identifying essential mechanisms needed to satisfy the Service Discovery Principles.

Future work includes investigating the non-functional aspects of service discovery such as time constraints, by simulation.

7. Acknowledgement

This research is sponsored by the Netherlands Organization for Scientific Research (NWO) under grant number 612.060.111, and by the IBM Equinox program. We thank K.Mills and C.Dabrowski from the National Institute of Standards and Technology (NIST), Gaithersburg, MD for their contributions to this work.

References

[1] K. Arnold, R. Scheifler, J. Waldo, B. O'Sullivan, and A. Wollrath. *The Jini Specification, V1.1*, 1999.

[2] C. Bettstetter and C. Renner. A comparison of service discovery protocols and implementation of the service location protocol. In *Proceedings of 6th EUNICE Open European Summer School: Innovative Internet Applications*, September 2000.

[3] D. Bosnacki. Implementing discrete time in promela and spin. In *Proceedings of the VIII Conference on Logic and Computer Science, LIRA '97,*, pages 25–32, 1997.

[4] C. Dabrowski, K. Mills, and S. Quirolgico. *A Model-based Analysis of First-Generation Service Discovery Systems*. Special Publication 500-260, National Institute of Standards and Technology, 2005.

[5] G.J.Holzmann. An analysis of bitstate hashing. In *Protocol Specification, Testing and Verification. 15th International Conference*. Kluwer Academic Publishers, November 1998.

[6] G.J.Holzmann. The model checker spin, primer and reference manual. Addison-Wesley, September 2003.

[7] E. Guttman, C. Perkins, J. C. Veizades, and M. Day. *Service Location Protocol, V.2*, December 2003.

[8] M. Huth and M. Ryan. Logic in computer science: Modelling and reasoning about systems. Cambridge University Press, First Edition, January 2000.

[9] Microsoft. *Universal Plug and Play Architecture, V1.0*, Jun 2000.

[10] V. Sundramoorthy, J. Scholten, P. G. Jansen, and P. H. Hartel. Service discovery at home. In *4th Int. Conf. on Information, Communications & Signal Processing and 4th IEEE Pacific-Rim Conf. On Multimedia (ICICS/PCM)*. IEEE Computer Society Press, December 2003.