

INFORMATION SOCIETY TECHNOLOGIES (IST) PROGRAMME

Project n°: FP6-IST-033563



SMEPP

Secure Middleware for Embedded Peer-to-Peer systems

WP5.0

D5.0.2 Implementation Progress Report

Author(s):	J. Serrano (TEC), S. Tillich (TUG), M. Diaz (UMA)
Status -Version:	Final - 1.0
Date:	25 June 2008
Distribution – Confidentiality:	Public
Code:	D5.0.2-ImplementationProgressReport.doc

Disclaimer

This document contains material, which is the copyright of certain SMEPP contractors, and may not be reproduced or copied without permission. All SMEPP consortium partners have agreed to the full publication of this document. The commercial use of any information contained in this document may require a license from the proprietor of that information.

The SMEPP Consortium consists of the following companies:

Participant no.	Participant name	Participant short name	Country
1 (Co-ordinator)	Universidad de Málaga	UMA	Spain
2	Tecnatom, S. A.	TEC	Spain
3	Technische Universität Graz	TUG	Austria
4	Siemens AG	SIEM	Germany
5	Valtion Teknillinen Tutkimuskeskus	VTT	Finland
6	Università di Pisa	UPI	Italy
7	Telefónica I+D	TID	Spain
8	Institute for Infocomm Research	I2R	Singapore

The information in this document is provided “as is” and no guarantee or warranty is given that the information is fit for any particular purpose. The user thereof uses the information at its sole risk and liability.

Document Revision History

Date	Issue	Author/Editor/Contributor	Summary of main changes
25 Sep 2007	0.0	José Serrano (TEC)	Initial draft
3 Oct 2007	0.1	Jose Serrano (TEC) Stefan Tillich (TUG)	Rearrangement of actual effort section. Description of progress in WP5.1
5 Oct 2007	0.2	Manuel Diaz (UMA)	Description of progress in WO5.2
4 April 2008	1.0	José Serrano (TEC)	Second release: First draft
25 June 2008	1.1	José Serrano (TEC) D. Garrido, R. Roman (UMA) F. Benigni (UPI) E. Hess (SIE) T. Paaso (VTT) J. Zhou (I2R) U. Payer (TUG)	Partner Inputs

Table of Contents

1	Introduction	5
1.1	Purpose.....	5
1.2	Audience	5
2	Scope	6
3	Expected and actual progress	7
3.1	Expected Progress.....	7
3.2	Actual Progress	7
3.2.1	WP5 Implementation Coordination	8
3.2.2	WP5.1 Security Services Implementation	8
3.2.3	WP5.2 Middleware Framework Implementation	14
4	Version Plan	18
4.1	Milestones.....	18
4.1.1	Versions	18

1 Introduction

1.1 Purpose

Implementation inside SMEPP project is divided in three work-packages:

- 5.1 Security Services Implementation
- 5.2 Middleware Framework Implementation
- 5.3 Network Specific Protocols

As stated in the Description of Work, work package 5 (Implementation Coordination) is responsible for monitoring the work plans for each implementation WP.

A new version of Deliverable 5.0.2 is released every six months. This version describes the progress of implementation from month 12 to month 18.

1.2 Audience

The audience for this document includes the project management, system analysts, system designers and testers.

2 Scope

This report covers the implementation progress for all the software development produced from month 12 until month 18 in the SMEPP work-packages 5.1 (Security Services Implementation), 5.2 (Middleware Framework Implementation) and 5.3 (Network Specific Protocols and Infrastructure)

3 Expected and actual progress

3.1 *Expected Progress*

According to the plans described in the Description of Work and the Initial Version Plan described in deliverable 5.0.1 (Configuration Management Plan) the implementation should be working towards the milestone of version 0.0, expected for month 18. This version should include Basic Component Infrastructure: Instantiation and binding, Communication, Discovery, Self-configuration and run-time adaptation.

According to the Description of Work, most tasks related to Implementation should have started. Namely,

- WP5
 - Task 5.1 Configuration Management and Version Planning
 - Task 5.2 Middleware Implementation Coordination
- WP5.1
 - Task 5.1.1 Secure Instruction Set Extensions
 - Task 5.1.2 Secure HW modules for asymmetric cryptography
 - Task 5.1.3 Design of new asymmetric cryptography primitives
 - Task 5.1.4 Implementation of symmetric and asymmetric algorithms
 - Task 5.1.5 Elliptic curve construction algorithm
- WP5.2
 - Task 5.2.1 Basic Component Infrastructure Development
 - Task 5.2.2 Component and application Development Tools
 - Task 5.2.3 Extra functional Properties Support
- WP5.3
 - Task 5.3.1 Analysis of Specific Protocols and Infrastructures
 - Task 5.3.2 Evaluation of Specific Protocols

3.2 *Actual Progress*

At end of month 18, the following tasks had started:

- WP5 Implementation Coordination

- Task 5.0.1 Configuration Management and Version Planning
- Task 5.0.2 Middleware Implementation Coordination
- WP5.1 Security Services Implementation
 - Task 5.1.1 Secure Instruction Set extensions for EP2P lightweight devices
 - Task 5.1.3 Design of new asymmetric cryptographic primitives
 - Task 5.1.4 Implementation of Symmetric and asymmetric algorithms
 - Task 5.1.5 Elliptic curve construction algorithm
 - Task 5.1.6 Implementation of Security Protocols
- WP5.2 Middleware Framework Implementation
 - Task 5.2.1 Basic Component Infrastructure Development
 - Task 5.2.2 Component and application Development Tools
 - Task 5.2.3 Extra functional Properties Support
- WP5.3
 - Task 5.3.1 Analysis of Specific Protocols and Infrastructures
 - Task 5.3.2 Evaluation of Specific Protocols

Description of actual progress in each work-package is described below:

3.2.1 WP5 Implementation Coordination

3.2.1.1 Task 5.1 Configuration Management & Version Planning

Due to delays in the architecture work-package, the version plan has to be updated. The new version plan is shown in this document.

3.2.1.2 Task 5.2 Middleware Implementation Coordination

Implementation status was reviewed at Malaga meeting (January 2008). As there is some delay in the architecture definition, middleware implementation is focusing in development tools and the underlying runtime framework.

3.2.2 WP5.1 Security Services Implementation

Siemens: The Siemens contributions to WP 5.1 comprises work for the tasks 5.1.3 “Design of new asymmetric primitives” and 5.1.5 “Elliptic curve construction algorithm”.

Concerning task 5.1.1, we focused on the development of side-channel-resistant algorithms for scalar multiplication on elliptic curve defined over finite fields. The goal was to find methods that achieve sufficient performance on computationally weak platforms used in sensor motes. (A typical example of such a platform is the Atmega128 device which is contained the MICAZ wireless sensor mote.) To this, we developed implementations of finite field arithmetic (in C and partially also in the specific assembler language) for finite fields of characteristic 2 and for extension fields of type $GF(p^k)$ and realized various scalar multiplication techniques. The used elliptic curves (over characteristic 2 fields and over extension fields $GF(p^k)$) were chosen in such a way that they provide a similar level of security. It turned out that the achievable performance depends widely on the chosen platform. In case of the Atmega128 microcontroller, finite fields of characteristic 2 are the best choice to realize an elliptic curve based cryptosystem.

As for task 5.1.5, we completed the implementation of an point counting algorithm (first version) for optimal extension fields $GF(p^k)$, $p < 255$. The algorithm makes it possible to construct cryptographically strong elliptic curve within one hour on a standard PC. Further, we developed a new “early-abort strategy” to discard curves with weak parameters at a very early phase of the program run. This new approach accelerates the elliptic curve construction by the factor 30. The new method supports heavily the construction of elliptic curves with special additional requirements that are necessary for elliptic curve cryptosystems, where very costly operations like field inversions are dismissed.

UMA-SEC: In this WP, we have been focusing on the development of situation awareness mechanisms for highly constrained platforms (i.e. sensor networks). Self-configurability is an essential property for distributed systems such as sensor networks. For example, a routing protocol needs to discover the state of the neighborhood in order to reconfigure and adapt itself to the changing environment. In order to achieve this autonomy, it is required to develop certain mechanisms that allow a certain node to discover the events that are happening on its surroundings and react to them. These mechanisms are known as situation awareness mechanisms.

The development of these awareness mechanisms is based on the intrusion detection systems studied on WP4.2. First, we enumerated the possible events (caused either by malfunctioning components or by malicious entities) that can happen in a distributed system, and later we identified the collateral effects that are caused by such events. By analyzing these collateral effects, it is possible to infer the existence of events and provide such knowledge to other subsystems such as the routing protocols.

Note that the development of these mechanisms within WP5.1 was not initially considered explicitly in the work plan, but they were identified as an essential part of the security architecture of the middleware (as a enabler of the self-configuration property) in WP4. Therefore, this development is being carried out on this WP as part of task 5.1.4 due to its main focus on the development of security mechanisms that will support other parts of the middleware (as indicated on the objectives of WP5.1, “This WP will be in charge of the implementation of the security infrastructure [...]”).

TUG: This WP is the core of the spiral model and was made up as a Java prototype to be able to experiment with different routing approaches. As a starting point (for our experiments), ARIADNE was chosen, which is a demand routing protocol for ad-hoc networks. ARIADNE seemed to be appropriate, since a very simple routing mechanism (DSR) was combined with a very efficient authentication mechanism (TESLA). The authentication mechanism has also been shown to be implementable on a sensor networks. This is the smallest platform we want to support. Additional to the basic mechanisms provided by ARIADNE, we introduced group management mechanisms to meet all SMEPP-requirements. At the moment just one base group can be generated, but multi-base groups will be supported soon.

The initial authentication phase is fully implemented (RSA), as well as the subsequent generation and exchange of session keys and authentication of subsequent communication messages.

I2R: During the past 6 months, I2R has implemented the following as part of the contribution to SMEPP;

- Enhanced Stateful Public Key Encryption (Version 1 completed)
- Key Drifting Protocol (Version 1 completed)
- ID Based Signature Scheme (Currently under development phase)
- Crypto Library

Enhanced Stateful Public Key Encryption (E-SPKE)

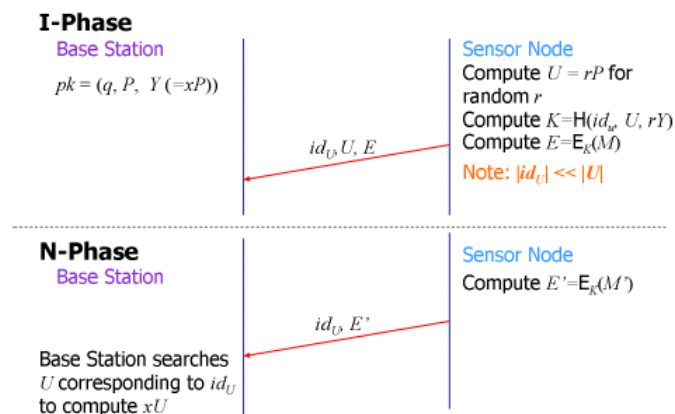


Figure 1. Enhanced Stateful Public Key Encryption Protocol

ESPKE is designed to provide data confidentiality during the transmission of the data over the air in a single hop environment. In our protocol, authentication of the forwarded public key is assumed. The ESPKE scheme is divided into 2 different phases. The I-Phase is the key setup phase whereby the nodes will refresh its public key with the base station. Base station will also store the node's public key together with the index (which is also the improvement over SPKE). This index will be use by the node for its subsequent message

transmission until the next key refreshment. This public key will be used to generate the session key similarly to the DH key exchange protocol. (Base station public key is pre-deployed in the node itself) The next phase is the communication phase or N-Phase. In this phase, the node will piggy backed the index together with ciphertext that is encrypted by the session key. Upon receiving of an incoming message, the base station will retrieve the corresponding public key based on the index and generate the session key that will be used to decrypt the ciphertext.

In the development of our ESPKE protocol the following cryptographic primitives were used:

- Elliptic Curve Cryptography (ECC) [5][6]: Allows smaller key size and lower computation overhead as compared to other public key cryptography like RSA [2]. For example ECC-160 offers similar security level as compared to RSA-1024. Hence ECC is suitable for offering security in a constraint environment like WSN. In our development, we have adopted the TinyECC [7] security library that was developed by NCSU.
- Hybrid Encryption: As public key encryption is always slower than its symmetric key based counterpart, we have adopted the hybrid encryption scheme in the ESPKE protocol. In the hybrid scheme, the individual public key, based on ECC, is shared between 2 communicating parties. This public key is then used by the individual to generate the shared symmetric key (session key) which will be used to encrypt the transmitted data under the “Data Encapsulation Mechanism (DEM)” [8].
- Diffie-Hellman Integrated Encryption Scheme (DHIES): Stateful DHIES [9] improved the efficiency of the original DHIES scheme through the use of “states” maintained by each individual party. This reduces the computation load for the regeneration of the symmetric key each time a message needs to be encrypted. Our ESPKE protocol further improved on the Stateful PKE [4] by reducing the transmitted overhead through the use of indexing.
- Advanced Encryption Standard (AES): AES-128 is the (128 bits) block cipher used in our development to encrypt the transmitted data using the symmetric key generated by ESPKE. This will provide the data confidentiality of the transmitted data. For our ESPKE, the AES-128 block cipher was designed and developed by TUG.
- Hash Message Authentication Code (HMAC): In our crypto-library, we have also developed HMAC based on SHA-1 algorithm to be used by either the sensor node or PC for message integrity protection.

The initial prototype for E-SPKE has already been developed on both the MicaZ (nesC) and PC (Java), and we are able to provide key refreshment, data confidentiality/integrity from the wireless sensor nodes (MicaZ) to the base station (PC). Currently authentication is not provided in the scheme and the next version of this protocol will be implemented to include a signature scheme. This

is to provide authentication for the keys that are sent over the air during the key refreshment period. (Analysis for this protocol can be found in D4.4)

Key Drifting Protocol (KDP)

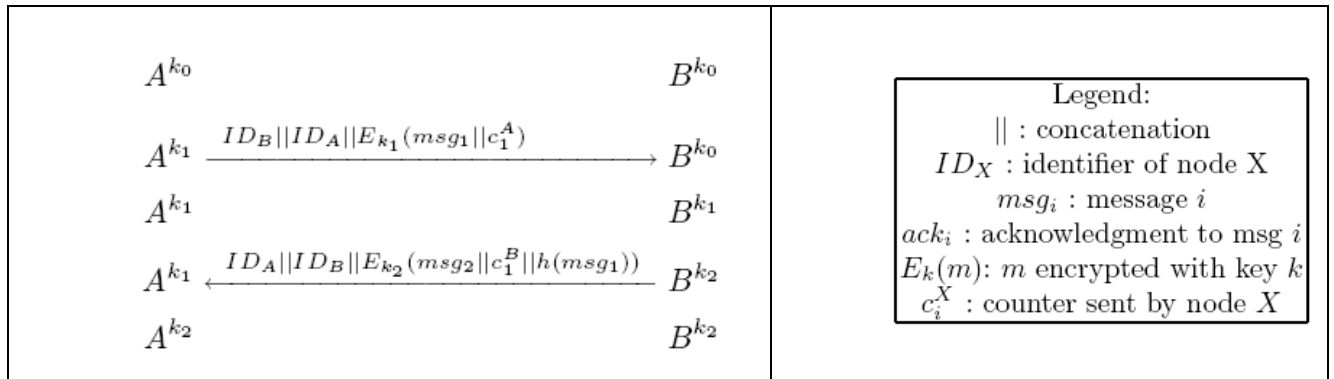


Figure 2: Key Drifting Protocol

The Key Divergent Protocol is able to enhance the security protection for existing system and incur low extra communication overhead as compare to existing key update acknowledgement system. The KDP is easy to integrate to any existing system using symmetric key encryption. It is almost transparent to the system. The only difference between ordinary system and KDP is: we need a little time to do the “brute force attack” on key recovery. Since we always start brute force attack on last key, so if the keys not changed, only one decryption is necessary. As KDP updates its key every time during communication, the attacker will not be able to discover the key if not monitoring the network at all time.

When the KDP used with any random key pre-distribution scheme, it is able to resist the node capture in that communication link. Only the compromised node will be affected, other remains unaffected. Because the keys shared between nodes will be completely different after enough time passed. Thus the KDP is able to provide an extra security while minimize the energy consumption of sensor mote during communication. This is especially significant in sensor network with limited resource of power.

The current implementation of the KDP is developed only on MicaZ (TinyOS, nesC) and uses AES-128 for its encryption/decryption.

ID based Signature Scheme

ID Base Online/Offline Signature (IBS) was particularly design to suit Wireless sensor network. IBS enjoys significant improvement in computational and storage cost over existing digital signature scheme. Differences between IBS and existing scheme are: IBS allow signer (sensor) to reuse the offline pre-computed information multi-times, in contrast to one-time usage in existing scheme. IBS is dividing into two portions, base station and mobile station. Base station will do the Extract, Offline sign and Verification of the signature. Mobile station just did the online sign with the message.

First, all the parameter will be decide by the base station. After that, base station will do the extract and offline sign. After that, base station will download this pre-compute information into mobile station. Finally, sleep and wait for verify incoming data.

When mobile station want to transmit data to base station, first it will generate a random value, y , and use it to sign with the message to be send to base station by using pre-compute information from base station.

Note that the pre-compute information is unique to each mobile station and contain the identity of each mobile station, hence during verification of signature, base station just need to compare the identity of the base station and it will know the signature belong to who and valid or not.

We are currently in the midst of implementing this signature scheme. Modules that have been implemented include:

- Offline signing and verification on Windows (PC, Java)
- Online signing on TinyOS (MicaZ, NesC)

Crypto Library

Other than the 3 protocols discussed above, we have also developed a crypto library suitable to be used on TinyOS platform developed in NesC language. The summary of the crypto library is shown in table 1.

Functions	Platforms / Language	Contributions	Limitations / Future Works
AES -128	TinyOS (NesC)	TUG – Assembly Codes for AES I2R – NesC wrapper interface and various modes for block ciphers	- ECB, CBC, OFB, CFB modes are implemented
AES – 128 AES – 192 AES – 256	Windows (C/C++/Java), TinyOS (NesC)	Shammus Software – Miracl I2R – Converting Miracl AES implementation to NesC	Currently support ECB, CBC, CFB and OFB unable to use both the decrypt and encrypt functions on the same sensor mote Currently we are using Bouncy Castle’s AES implementation as it is found to be faster than Miracl for smaller data size (higher overheads to call Miracl’s AES implementation by JNI)

Functions	Platforms / Language	Contributions	Limitations / Future Works
OneTimePad	Windows (C / Java), TinyOS (NesC)	I2R- kind of a OneTimePad for encryption and decryption on the sensor notes	
SHA – 1 SHA – 256 SHA – 512	Windows (C/C++/Java), TinyOS (NesC)	Shammus Software – Miracl I2R – Converting Miracl SHA implementation to NesC	
HMAC – SHA1	Windows (Java/C++), TinyOS (NesC)	I2R – based on SHA functions in Miracl	- Other modes of HMAC to be implemented (HMAC – SHA256...)
RNG	TinyOS (NesC)	I2R- seed value of RNG based on battery level (Voltage)	

Table 1. Summary of Crypto Library

3.2.3 WP5.2 Middleware Framework Implementation

UMA: The SMCOM-RCF is a Java implementation of the SMCOM component model. It has been developed using Java 1.6 SE, and actually is been ported to Java ME over PDAs. The current version is provided in a .jar file, and must be included into the projects that want to define SMEPP Components to develop applications. There are two main packages:

- SMCOM-RCF.Kernel: contains the core functionality of the runtime environment, including scheduler thread and data structures.
- SMCOM-RCF.SMEPPComponent: contains the definition of user classes, like SMEPPComponent (active and passive) SMEPPEvent and ISMEPPInterface.

Current version allows the next features:

- Defining SMEPPComponents.
- Defining ISMEPPInterfaces.
- Defining SMEPPEvents.
- Interconnect SMEPPComponents.
- Synchronize components using wait and call operations over defined interfaces.
- Subscribe to events (or wait synchronously).

- Raise events.

Future work includes:

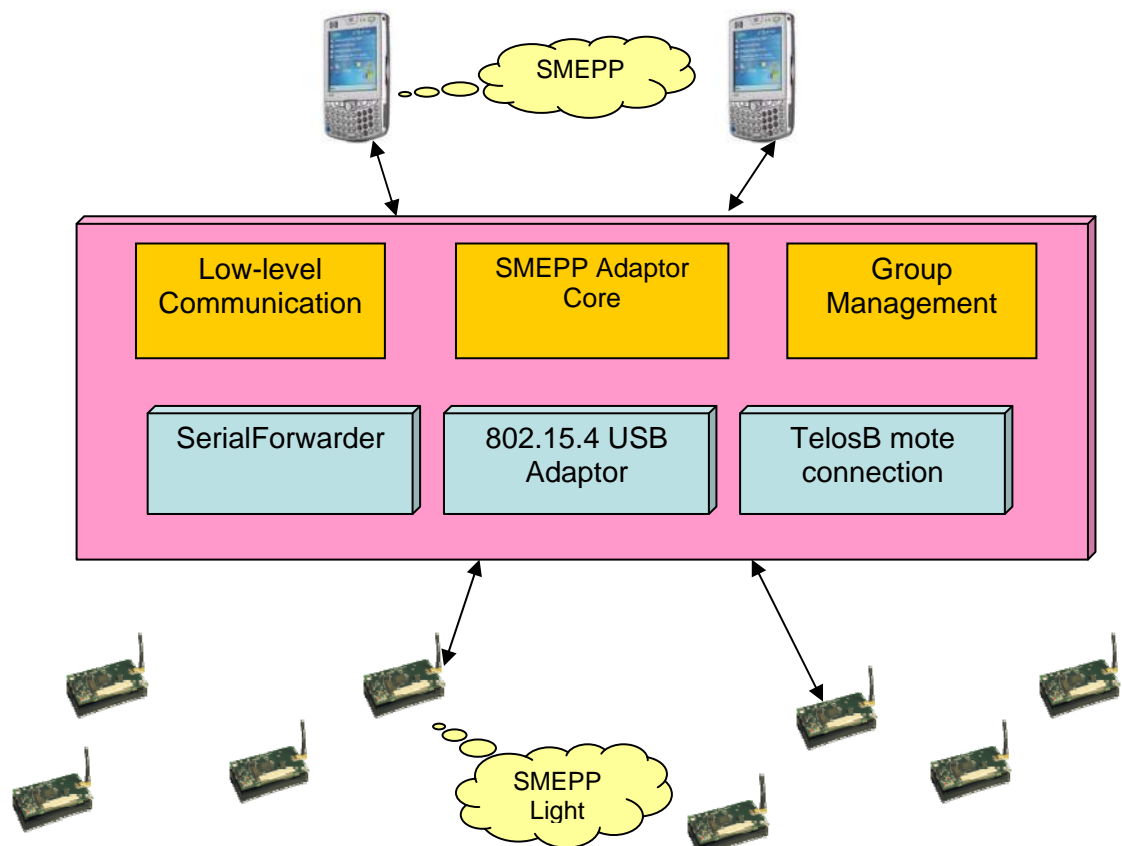
- Timeout versions of wait and call operations.
- Exception handling.
- Run the RCF over sensors supporting Java (like SUNSpot).

Current version is being used in the development of SMEPP Middleware.

SMEPP Light Adaptor prototype:

SMEPP Light Middleware is a modified version of SMEPP where services are not present and the basic way of communication is event-based. For this reason, the interaction between SMEPP versions is mandatory; this way a device that uses SMEPP (e.g. PDA or mobile phone) should be able of joining to SMEPP Light groups and subscribe to its events.

UMA is currently developing an adaptor in order to enable full communication between SMEPP and SMEPP Light middleware. In the following figure is shown the conceptual architecture of the adaptor.



The adaptor is based on a tool that makes visible the SMEPP Light groups to SMEPP devices by means of *SMEPP Light Group Proxy* that gives a representation of a group created by sensors. Obviously, per each group the

proxy creates a general service that contains the events provided by the sensors inside the group.

Other point that must be taken into account is the way that we can communicate the WSN with other devices. This aspect is important due to the impossibility of communication directly between motes and other devices, the ones use 802.15.4 wireless standard while standard devices use WiFi 802.11. Fortunately, there are different solutions, using a Stargate gateway, or 802.15.4 USB adaptors, or even using TelosB motes that are equipped with USB connector. The adaptor will be divided so the user just have to change one component to adapt to the lower level communication.

SecureLime-based service model prototype:

UPI have developed a proof-of-concept prototype implementation of the SMEPP model that can be used to test/simulate interactions of peers and services. This prototype is mainly intended as a preliminary study to verify the effectiveness of the service model defined in D2.1, however it also offered the opportunity to implement a SMoL2Java translator that can be reused for the development of SMEPP supporting tools.

The prototype is built upon a shared tuple spaces data abstraction offered by SecureLime, an extension of the Lime coordination language, that adds security properties to tuples and (federated) tuple spaces. The generative communication featured by tuple spaces can be seen as an enhancement of the basic coordination mechanism offered by standard data-centric storage techniques (e.g., distributed hash tables), which will be basis of the final SMEPP middleware.

To deal with SMoL specifications of the behaviour of services and peers, we have implemented a SMoL2Java translator (which generates Java code from a SMoL specification) that produces executable Java code that runs on top of SecureLime. The translator can be reused to realize tools that automatically generate java code that runs on top of the actual SMEPP APIs.

The main limitations with respect to the service model is that the prototype only features a basic mechanism for the discovery of service contracts based on the syntactic matching of tuples. As regards security mechanisms, one of the limits of the prototype is that it does not provide a way to change "on-the-fly" the tuple space passwords (i.e. the network/group keys). The prototype is freely downloadable from <http://www.smepp.org/Downloads.aspx>.

SMoL simulator:

UPI have developed an interactive tool capable of simulating every possible evolution of a SMEPP entity specified by a SMoL description. This SMoL simulator is a sort of workflow engine capable of generating, loading and saving (exporting in XML format) execution traces of SMEPP peers and services whose behaviour is specified using SMoL language, thus it can be used by SMEPP developers during development, maintenance and debug of SMEPP applications presenting (or generated from) a SMoL specification. Each

execution trace can be interactively guided and monitored through a graphical user interface, or automatically generated for further analysis.

The SMoL simulator is written in C#.

4 Version Plan

As mentioned in the DoW, the project will follow an iterative approach and will therefore produce incremental versions of the middleware in different iterations.

Deliverable 5.0.1 “Configuration Management System” defined the initial version plan. This version plan has been modified in this report to reflect the current situation of the project.

Version 0.0. has been delayed from month 18 to month 22.

Version 0.1 has been delayed from month 24 to month 28.

Version 0.2 has been delayed from month 30 to month 32.

4.1 Milestones

The main milestones in the different sub-packages are:

- (WP5.2) M18: First version of the basic infrastructure implementation. Delayed to month 22.
- (WP5.2) M24: Component Infrastructure & Tools Implementation
- (WP5.2) M24: First version with extra functional support
- (WP5.1) M24: Cryptographic algorithms and protocols implemented and security support HW module design
- (WP5.2) M30: Upgrading and Extension Support implemented
- (WP5.3) M30: Network and device specific protocols implementation and integration

4.1.1 Versions

With these milestones in mind, three main versions are defined:

- Version 0.0 (M22). Includes:
 - Basic Component Infrastructure: Instantiation and binding, Communication, Discovery, Self-configuration and run-time adaptation
- Version 0.1 (M28). Includes:
 - Component Infrastructure & Tools
 - Extra functional properties support: Integration of security services & network QoS.

- Cryptographic algorithms and protocols
- Version 0.2 (M32). Includes:
 - Upgrading and extension support
 - Network and device specific protocols

After version 0.2 has been released, other new versions will appear with the feedback and issues from the application.