SANDIA REPORT
SAND2009-6307
ORNL/TM-2009/205
Unlimited Release
Printed October 2009

Coupling a Transient Solvent Extraction Module with the Separations and Safeguards Performance Model

Benjamin B. Cipiti, Valmor F. de Almeida, Ian C. Gauld, Joseph F. Birdwell, and David W. DePaoli

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550

Sandia is a multiprogram laboratory operated by Sandia Corporation,
a Lockheed Martin Company, for the United States Department of Energy's
National Nuclear Security Administration under Contract DE-AC04-94AL85000.

Sandia National Laboratories

SAND2009-6307
ORNL/TM-2009/205
Unlimited Release
Printed October 2009

Coupling a Transient Solvent Extraction Module with the Separations and Safeguards Performance Model

Benjamin B. Cipiti
Advanced Nuclear Fuel Cycle Technology
Sandia National Laboratory
P.O. Box 5800
Albuquerque, NM 87185-0747

Valmor F. de Almeida, Ian C. Gauld, Joseph F. Birdwell, and David W. DePaoli
Oak Ridge National Laboratory
1 Bethel Valley Road
P.O. Box 2008
Oak Ridge, TN 37831-6181

Abstract

A number of codes have been developed in the past for safeguards analysis, but many are dated, and no single code is able to cover all aspects of materials accountancy, process monitoring, and diversion scenario analysis. The purpose of this work was to integrate a transient solvent extraction simulation module developed at Oak Ridge National Laboratory, with the Separations and Safeguards Performance Model (SSPM), developed at Sandia National Laboratory, as a first step toward creating a more versatile design and evaluation tool. The SSPM was designed for materials accountancy and process monitoring analyses, but previous versions of the code have included limited detail on the chemical processes, including chemical separations. The transient solvent extraction model is based on the ORNL SEPHIS code approach to consider solute build up in a bank of contactors in the PUREX process. Combined, these capabilities yield a more robust transient separations and safeguards model for evaluating safeguards system design. This coupling and initial results are presented. In addition, some observations toward further enhancement of separations and safeguards modeling based on this effort are provided, including: items to be addressed in integrating legacy codes, additional improvements needed for a fully functional solvent extraction module, and recommendations for future integration of other chemical process modules.

Contents

Figures

Acronyms

AFCI	Advanced Fuel Cycle Initiative
DEPO	Design Evaluation Process Outline
I/O	Input/Output
MBA	Material Balance Area
MC&A	Material Control and Accountability
MS	Mass Spectrometer
NEAMS	Nuclear Energy Advanced Modeling and Simulation
ODE	Ordinary Differential Equation
PUREX	Plutonium and Uranium Extraction
SEPHIS	Solvent Extraction Process Having Interaction Solvents
SSPM	Separations & Safeguards Performance Model
UREX	Uranium Extraction

1.0 Introduction

The Safeguards Modeling and Simulation Expert Group was established in 2008 by NA-242 to identify simulation and modeling tools available for design and evaluation of safeguards systems for both the U.S. and Japanese nuclear energy programs. A number of codes were identified in that work, but a gap was found in the area of evaluation and analysis. Most of the codes were designed to look at a specific aspect of safeguards design or analysis, and many were dated. No one code was able to examine a full suite of safeguards analyses including materials accountancy, process monitoring, and diversion scenario analysis.

This work grew out of that task as a first step in developing a standardized code for safeguards analysis and evaluation. The Separations and Safeguards Performance Model (SSPM), developed at Sandia National Laboratory, was designed for materials accountancy and process monitoring analyses, but it only included limited detail about the chemical separations and other chemical processes, thus reducing its effectiveness for process monitoring. In particular the solvent extraction unit operation was assumed to be at steady-state at all times under a single chemical equilibrium condition for the entire bank of contactors. While that assumption allowed rapid calculations and provided reasonable estimates under steady-state conditions, it did not provide realistic treatment of transients in the operation. The purpose of this work was to illustrate the capability to improve safeguards models to estimate transient scenarios by introducing a simulation module that performs predictions of concentrations under varying conditions and provides a transient material balance for the solutes in the solvent extraction process. The solvent extraction module developed in this work used the distribution coefficients and mixer-settler contactor model of the SEPHIS (Solvent Extraction Process Having Interaction Solvents) code developed at ORNL [1,2].

While this current effort is focused on incorporation of a transient solvent extraction simulation capability into SSPM, the procedure can potentially be applied to integrate other codes as well. Additional capabilities could be integrated into this code using a similar approach. Although the current coupling would be considered export-controlled, versions could be developed that would be open for foreign use. Also, the SSPM could also be sent out in an open version that could be set up to integrate a proprietary code in a different country.

The Nuclear Energy Advanced Modeling and Simulation (NEAMS) campaign in the Advanced Fuel Cycle Initiative (AFCI) program is tasked with developing advanced models for fuel cycle facilities. The results provided here may provide guidance for NEAMS in regards to how the Safeguards by Design approach could be integrated in plant-level simulation codes. One of the advantages of this approach is the use of codes that have already been through the verification and validation process.

2.0 Background

2.1 Safeguards Simulation and Modeling Expert Group Analysis

The original goal of the Expert Group was to identify dedicated software and codes that are used by safeguards professionals. The group also discussed the value of coupling some of the codes into a more powerful tool with more capabilities. Part of this goal is driven by the Safeguards by Design concept that seeks to design safeguards aspects into a plant early in the design process rather than adding the system later. Safeguards by Design will be required to keep costs minimized given that regulations are constantly changing.

Within the Expert Group, a systems engineering process was proposed for providing a framework for design and evaluation of safeguards systems [3]. This process was modeled after the Design and Evaluation Process Outline (DEPO) methodology which has been applied for over 25 years for physical protection system. However, the approach was aimed specifically at safeguards. Figure 1 shows the process.

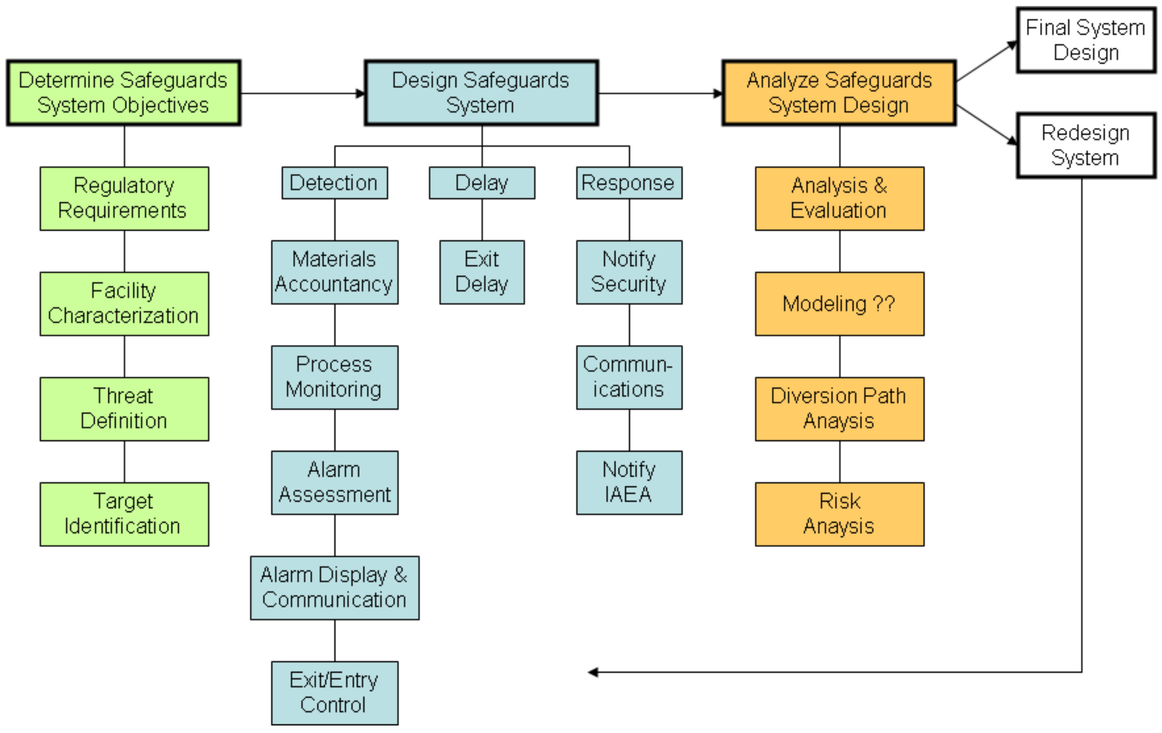


Figure 1: Systems Engineering Process for Safeguards Design

After examining the various codes and where they fit into the process, a gap was found in the Analysis and Evaluation section. Codes were found to exist, but all have shortcomings. SEPHIS was identified as a solvent extraction code that can play a role, and the SSPM is a materials accountancy and process monitoring model developed at Sandia. These existing codes provide a useful starting point for exploring a more versatile analysis tool.

2.2 SSPM

The Separations and Safeguards Performance Model (SSPM) [4] is a high-level materials tracking model of a reprocessing plant developed at Sandia for materials accountancy and process monitoring analysis. The SSPM was chosen since it is one of the more modern models and due to familiarity at Sandia. The SSPM is constructed in Matlab Simulink and tracks cold chemicals, bulk fluid flow, solids, and mass flow rates of elements 1-99 on the periodic table. However, since separations modeling was not the goal of the SSPM, the chemical processes were described with simplified models, and all separation efficiencies have been assumed with static values. The main purpose the model was to simulate materials accountancy and process monitoring measurements. This data is used to simulate inventory difference calculations and examine the instrumentation response to material loss scenarios.

Figure 2 shows the front end of the SSPM in the Simulink environment. The processing vessels are shown as the black rectangles and contain functionality that models their operation. Each signal connecting the blocks contains a 101-element array that keeps track of the mass flow rates of elements 1-99, the total liquid low rate, and the total solids flow rate.

The blue and green blocks, which may be connected to either process streams or vessel inventories, are used to simulate accountancy or process monitoring measurements. For example, the “Acc MS” block above the accountability tank simulated a plutonium concentration measurement from a sample taken once every 8 hours. Each measurement block is customized for the particular measurement. The red diversion block may be moved throughout the model to determine the instrumentation response to material loss. Elsewhere in the model (not shown) all of the data from the measurements are used to determine an overall inventory difference.

Figure 3 shows the separations portion of the model for a PUREX reprocessing plant. PUREX contains one reprocessing step to produce a U and Pu product stream, and is representative of reprocessing plants around the world.

In this particular example, mixer-settler contactors are used to carry out the solvent extraction operation. In the original SSPM, the contactor bank was modeled as a tank that produces product streams at equilibrium conditions across the entire bank of contactors. Therefore, the transient build-up of solutes along the contactor bank was not considered. The purpose of this investigation was to incorporate a transient model to account for the mass build up of solutes in the PUREX process, and to better represent time-dependent diversion scenarios for safeguards analyses.

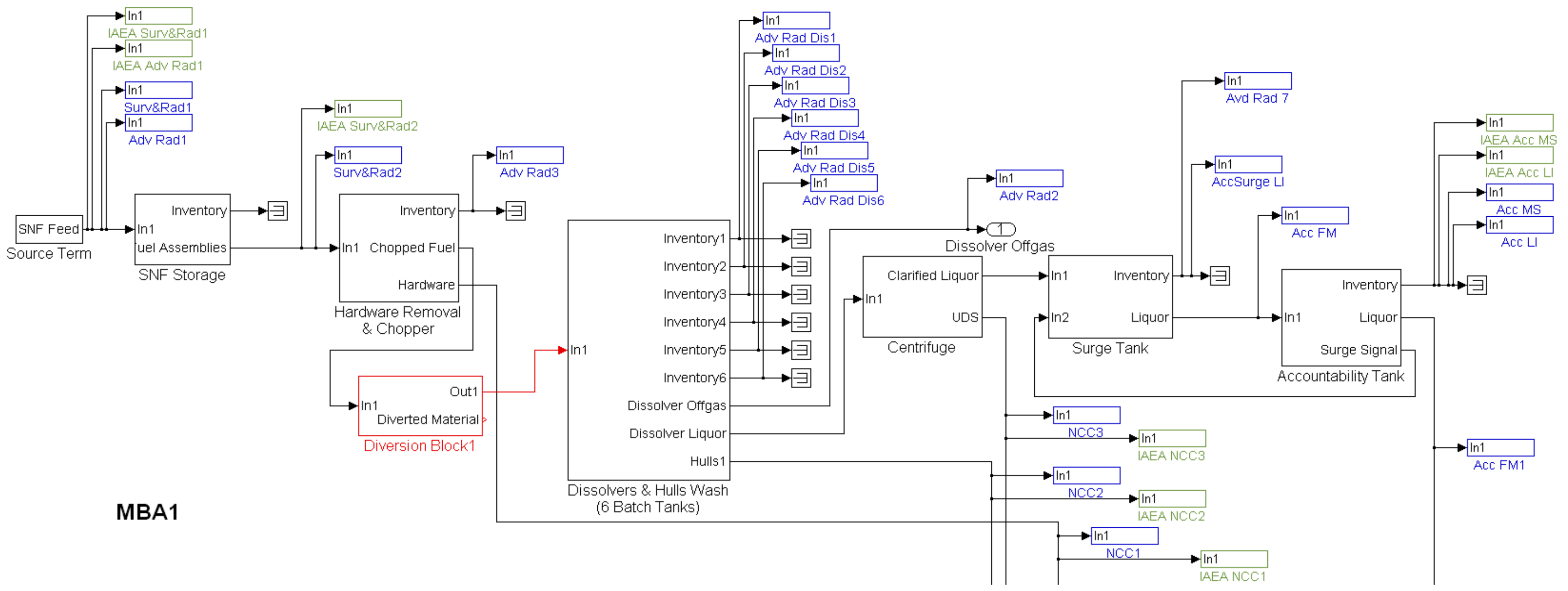


Figure 2: Front End (MBA1)

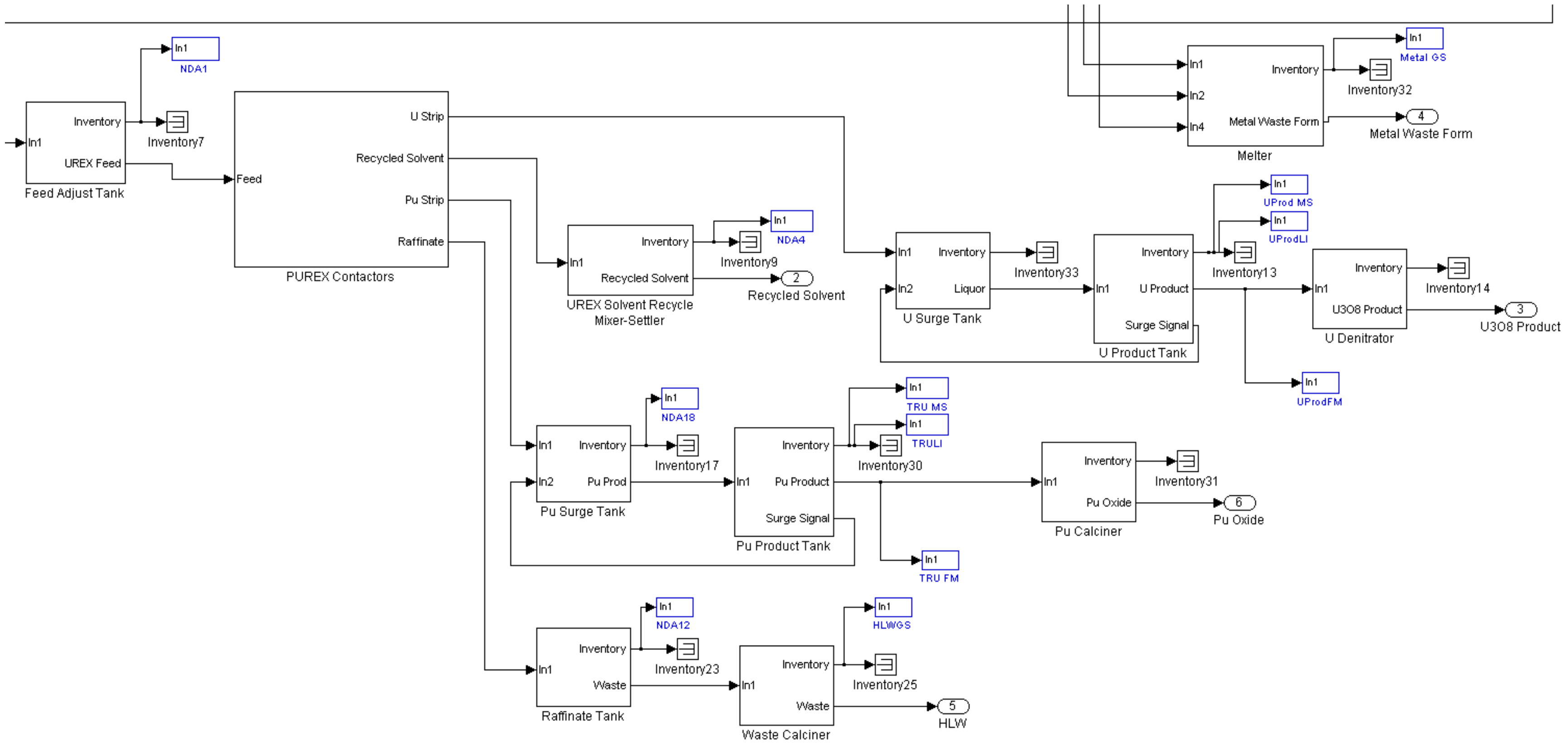


Figure 3: Separations (MBA2)

2.3 SEPHIS

This effort used a mathematical model based on the SEPHIS code to develop a solvent extraction module for integration with SSPM. SEPHIS (Solvent Extraction Process Having Interacting Solutes) is an existing process model that predicts solute concentration profiles in aqueous and organic phases throughout a solvent extraction process as a function of time. SEPHIS was chosen since it provides transient results. The SEPHIS model was originally designed from a simulation for the PUREX process based on an idealized model for mixer-settlers [5]. Further modifications to the program have led to simulations for THOREX, BUTEX, and a process for co-extraction of plutonium and neptunium [6]. SEPHIS has been used as the basis for PUREX flowsheet designs at Y-12, SRNL and Hanford.

The mathematical basis for recent versions was implemented in SEPHIS-MOD4 [7,8,9,10,11]. This model considered a number of solutes for the PUREX process, including nitric acid, uranium, plutonium (IV), plutonium (III), a plutonium reductant, and inextractable nitrates. The flow of solutes throughout the extraction process is modeled via ordinary differential equations. The distribution of solutes between the aqueous and organic phases is determined by subroutines that estimate distribution coefficients based on experimental data. Reduction reactions between solutes are also controlled in program subroutines. An idealized cascade of solvent extraction contactors modeled by SEPHIS is shown in Figure 4. In this figure, the contactors are mixer-

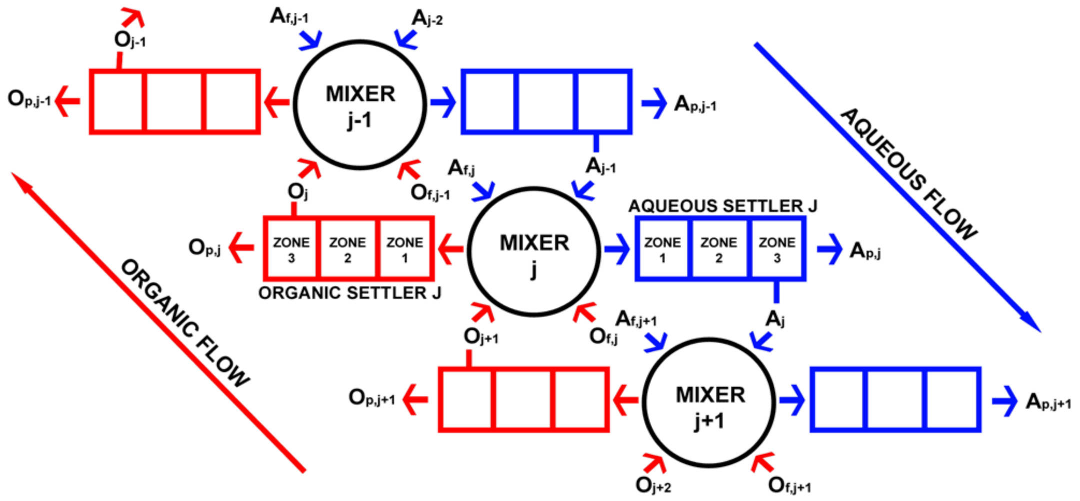


Figure 4: Overview of SEPHIS Model of Solvent Extraction Stages [8,9]

settlers; however, the model can be potentially applied to other contactors such as pulsed columns or centrifugal contactors through appropriate selection of parameters. The solutes enter each mixer through an aqueous or organic feed stream, the aqueous stream from the previous stage, and the organic stream from the next stage. Perfect mixing and equilibrium is assumed stage-wise; therefore, the aqueous and organic streams containing the corresponding compositions are defined by the distribution coefficients and are separated as they enter a respective settler. Upon leaving the settler, these streams can either leave the system or continue as interstage flow.

SEPHIS is designed to model transients in a bank of interconnected solvent extraction stages. The basic equation in SEPHIS is an unsteady-state mass balance of mixer contents. Between stage mixers are settlers, whose concentrations must also be solved. The settlers propagate outgoing concentrations from the mixer to the next stage or to a product stream; additional reactions may be taking place in the settlers such as plutonium reduction. The dynamics of transport through the system is governed by the residence time in each stage, directly related to the mixer and settler volumes and the fluid flow rates.

SEPHIS provides a knowledge base upon which to develop transient process models. The main program block performs process-generic mass balance calculations and iterations, while process-specific calculations (extraction coefficients, density changes, temperature effects) are performed in other code blocks. These coefficient values are, in all instances, calibrated by experimental data. Future needed work with SEPHIS includes improved treatment of kinetic processes and improved predictions of partitioning. However, the key relevant feature of SEPHIS as a solvent extraction process model is its ability to model the build up of solutes in a bank of ideal contactors as the transfer process approaches steady state and the transient effects of process upsets.

3.0 Integration of Transient Solvent Extraction in SSPM

The Simulink platform of the SSPM operates by signal propagation. The SSPM is a transient code, so it tracks all physical items with a coordinated time. Each flow stream is simulated with an array of signals that changes in time. A look at an individual element of the array will show the mass flow rate in that stream. The processing vessels use a combination of math functions to represent the physical function. For example, integration of the sum of the incoming and outgoing streams simulates a tank inventory.

The real power of Simulink is in the ability to use the Matlab workspace to calculate more advanced functions while still retaining the graphical interface of Simulink. Therefore a simple approach to couple codes via I/O is to write a Matlab function that writes input data to a file and makes a system call to an external code that uses the input data. This capability allowed the SSPM and the solvent extraction unit simulation module to share data.

3.1 Matlab Layers

Simulink has an embedded function block that was used for the integration. Any number of flow streams can enter and leave the block, but the block runs a short program to determine how the signals will be modified. Figure 5 shows the embedded function block that was used in the PUREX Contactor Block to call the external solvent extraction unit module.

The embedded function block can be thought of as including three layers under the PUREX contactors. The first layer is an enabled subsystem which must be present to only run the solvent extraction unit module periodically (for example, once every 15 minutes or once per hour). This helps to speed up the simulation time since otherwise the code would run on every simulation time step (further development of the solvent extraction unit will allow for small running times while still being called at arbitrarily small time steps).

The second layer is the embedded function that takes all the Simulink inputs and converts them into Matlab code. This code is shown in the Appendix. The only signal input from the model into this block is the actual feed flow rate into the contactor bank. The rest of the inputs shown in Figure 4 are user-entered variables that control the solvent extraction unit model. Most of these variables will be set at a default position, but they can be changed depending on the design of the contactor bank.

The third layer is a wrapper for the solvent extraction unit module. It is a simple interface written in the MATLAB language and stored into an “m” file. It requires a data structure (input) with variables that the SSPM needs to populate with values, and it provides a data structure (output) with fields that the SSPM requires. The wrapper creates an input data file for running the underlying external, previously compiled, solvent extraction simulation module via a system command, and builds the output data structure by reading the output file generated after the system command returns.

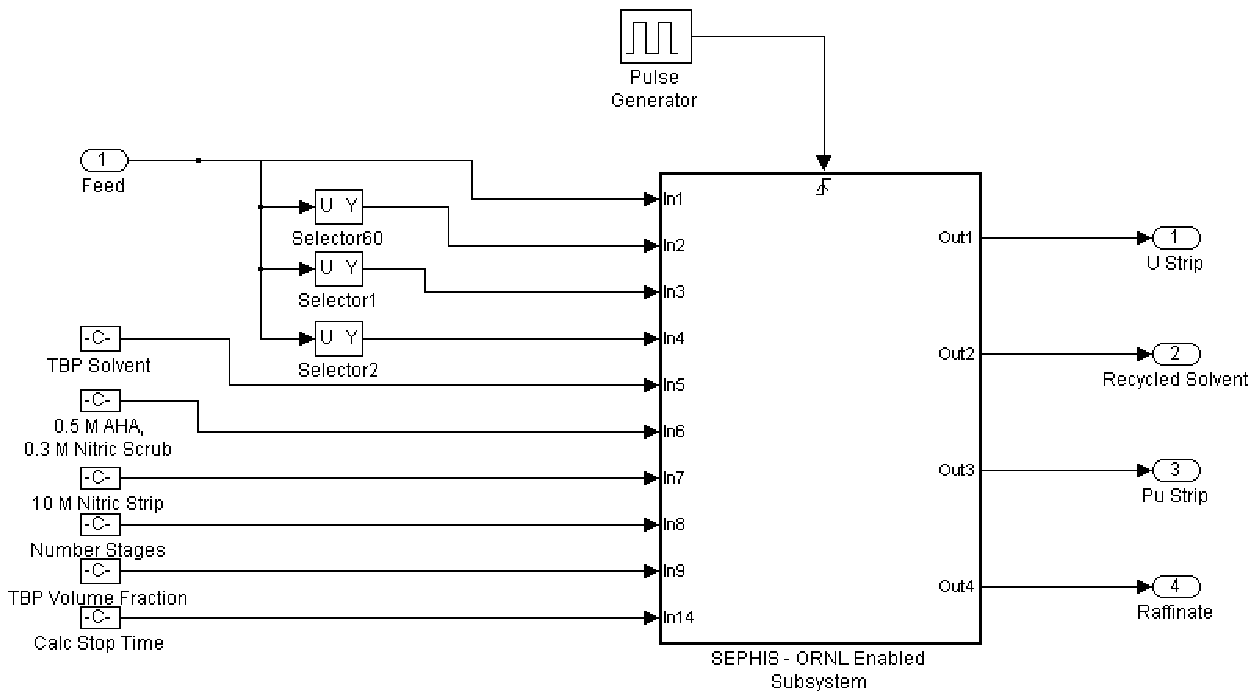


Figure 5: Embedded Function Block

3.2 Time Dependence

The SSPM runs with a variable time step in Simulink which means that the simulation time step can get larger or smaller depending on the amount of detail required. The time step varies depending on the system gradients. The simulation time is in units of hours, and the time step can be on the order of 0.001 hours (3.6 seconds).

If the solvent extraction module were called every time step, the simulation would slow down considerably, making the model more difficult to use on a desktop computer. A typical solvent extraction process does not need to be probed at intervals shorter than 0.1 minute (6 seconds) because the transients are much longer than this time interval. In test cases for this study, the solvent extraction module was called at intervals of 15 and 60 minutes wherein the state of the contactor bank was saved in between calls to be used at later times. Therefore this enables the operator to trade the determination of how long a change in the input variables will take to reach steady-state; this is a unique capability.

3.3 Embedded Function

The embedded function has one main purpose—it converts the Simulink signals into variables that can be used by the solvent extraction simulation module. The information from the feed signal that is extracted includes the feed volumetric flow rate, uranium mass flow rate, and plutonium mass flow rate.

The embedded code contains a script to only run the SolventExtractionUnit file periodically. SolventExtractionUnit is also an internal Matlab script, but it calls the external SEPHIS executable to generate results. It also adjusts for operation at startup or during continuous operation since the solvent extraction unit results depend on whether the contactors are at a cold start condition or not.

After the SolventExtractionUnit m-file runs, the embedded code takes the results and converts them back into Simulink outputs. These outputs are used to calculate the uranium and plutonium mass flow rates in each of the output blocks.

3.4 Solvent Extraction Unit

The SolventExtractionUnit “m” file is the interface to the solvent extraction module. It exchanges data with the embedded function through Matlab data structures and hides all the details of running the underlying solvent extraction simulation module. A system command is used by the “m” file to run the module which looks for an input file previously created by the interface and a state file created by a previous run which saves the contactor bank state from a previous time. After the system command returns, the wrapper function reads the output created by the module and passes the data back to the embedded function. The data contains volumetric flow rates of outgoing streams, and corresponding concentrations of main species; uranium, and plutonium.

The solvent extraction simulation module employed in this study was developed to provide the SSPM time-dependent concentration of solutes in the exiting streams of a bank of mixer-settler contactors used in the PUREX process. The solvent extraction model consists of three banks of 16 mixer-settlers (per bank), comprising U/Pu coextraction, U/Pu partitioning with Pu reduction, and U extraction for recovery of U and Pu from a fast breeder reactor spent fuel [8]. For a given input of feed, scrub, strip, back-scrub streams, and an initial state for all contactors, the module calculated the time-dependent variation of solute concentration in all internal and outgoing streams (product, organic waste, and raffinate); the time step used was 6 seconds. The model for the mass balance in each contactor followed the SEPHIS approach, and the distribution coefficients used were extracted from SEPHIS.

The initial condition for the system of contactors could be either a “cold-start” wherein the bank of contactors had no solute, or a previously computed state from an earlier run that was stored on disk. This allowed the calling module, SSPM, to execute the solvent extraction code for a given length of time and effectively suspend the solvent extraction simulation to be resumed at a later time. Since the transients in the solvent extraction process can be on the order of hours, this capability reduced the overhead of the calculation for the SSPM analysis which had the freedom to time step at a much smaller time scale.

The solvent extraction module interface was written in Matlab language and the data communication back and forth with SSPM was done through Matlab data structures. Internally the interface wrties input data to a file, and executes a system call for the solvent extraction module executable which writes the results to another file. Lastly, the interface reads the latter data and passes it back to the SSPM code.

It is recognized that there is significant room for continued improvement in solvent extraction modeling. In the development of the solvent extraction module, initial steps were taken to improve upon the SEPHIS mass balance approach, which assumes constant contact volumes and volumetric flow rates. In future efforts, this capability will be further developed to provide more realism on transients. In addition, a fully coupled system of ordinary differential equations will be solved, as opposed to the fragile, segregated approach used in SEPHIS; the solvent extraction module was organized to use robust ODE solvers currently available in standard scientific libraries. A review of the literature and possible collaboration with other DOE laboratories in the future will be sought to obtain distribution coefficients corresponding to comprehensive reaction mechanisms. Also, the current coupling of the module will be replaced via a modern approach of common components linked to a plant level framework; initial steps in that direction were also taken.

4.0 Modeling Results

Three test cases were carried out to illustrate the transient simulation capability. The results are only preliminary and considered work in progress.

The first test was to determine if the solvent extraction occurred as expected at startup and normal operation. Under the assumed plant configuration and operation, approximately 27 simulated hours pass at plant startup before the dissolver solution reaches the contactor bank in the solvent extraction unit. Once processing begins, at least a few simulated hours will pass before the contactors should reach a near steady-state condition. During these tests, one simulated hour runs in about 1 second on a standard desktop PC.

Figure 6 shows the result of the run wherein the solvent extraction unit was invoked at intervals of 60 minutes. Figures 6A and 6B shows the uranium and plutonium mass flow rates (kilogram/hour), respectively, entering the solvent extraction unit. Figures 6C and 6D show the uranium and plutonium mass flow rates leaving with the uranium product stream—these figures show that most of the uranium and essentially no plutonium goes into the uranium product (which is expected). Figures 6E and 6F show the uranium and plutonium mass flow rates leaving with the plutonium product stream. Again, the solvent extraction appears to be operating normally, with a small amount of uranium and all of the plutonium going into this product.

The plutonium mass flow rate at steady-state in 6B and 6F do not match up exactly, which does point to a problem in the current model. This discrepancy could be due to a number of issues that will need to be addressed in future work. The time step between SEPHIS runs could be partly to blame. Also, for this preliminary work, the SEPHIS results were calculated as ratios and applied to the SSPM flow rates. Both of these issues will need to be addressed in future versions.

The time lag shown in Figure 6 demonstrates that the coupling between the codes is operating as expected. Once uranium enters the contactors, it takes 4-5 hours before the uranium product reaches near steady-state. The same is true for plutonium. This time lag is due to material building up in the system, and the results when running the solvent extraction module stand-alone are the same.

The second test examined the effects of slight transients (in the form of a pulse) during normal operation. This test occurred from hours 60 to 100 and includes slight variations in the feed stream to the solvent extraction unit characteristic of normal plant transients that may be experienced (for example, when some of the unit operations are carried out in batch mode, or closing and opening of valves, or hold up and transients in pipes). Figure 7 shows the results. Again, the same 4-5 hours of lag before reaching steady-state is observed, and the separation of species occurs as expected. Without the SEPHIS integration, the original model assumed instantaneous separations of species.

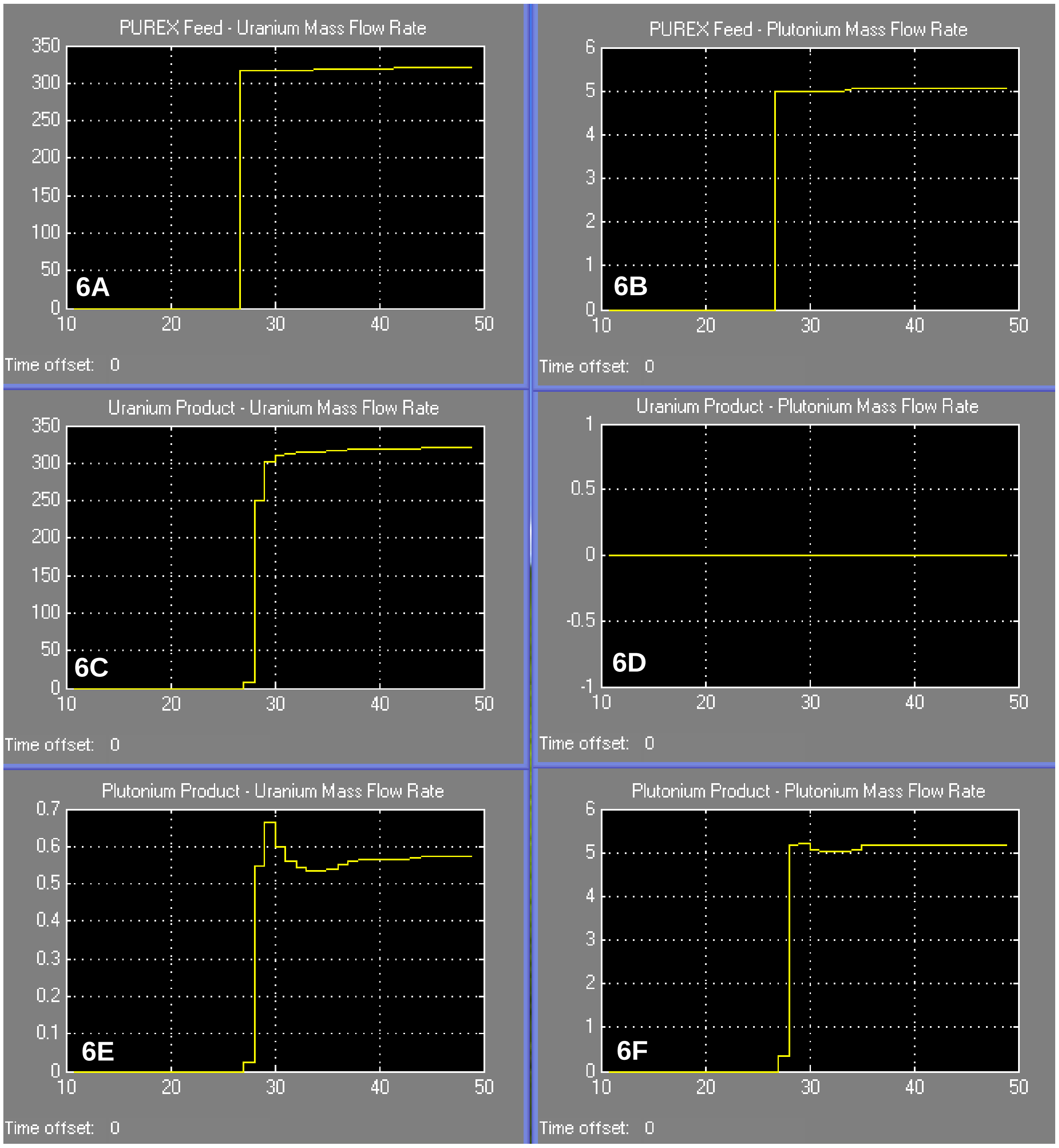


Figure 6: Startup and steady-state test. Mass flow rate in kilogram per hour

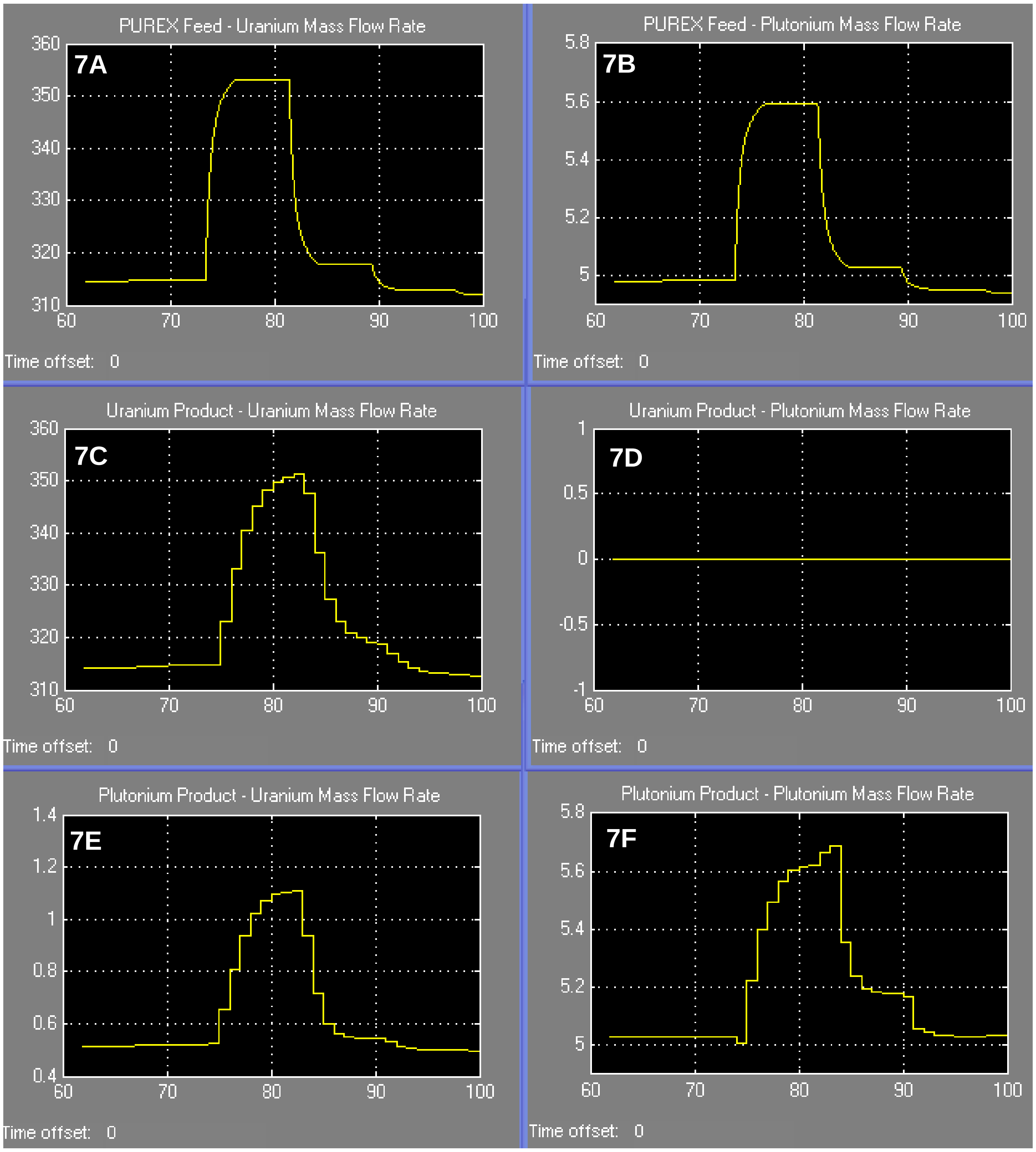
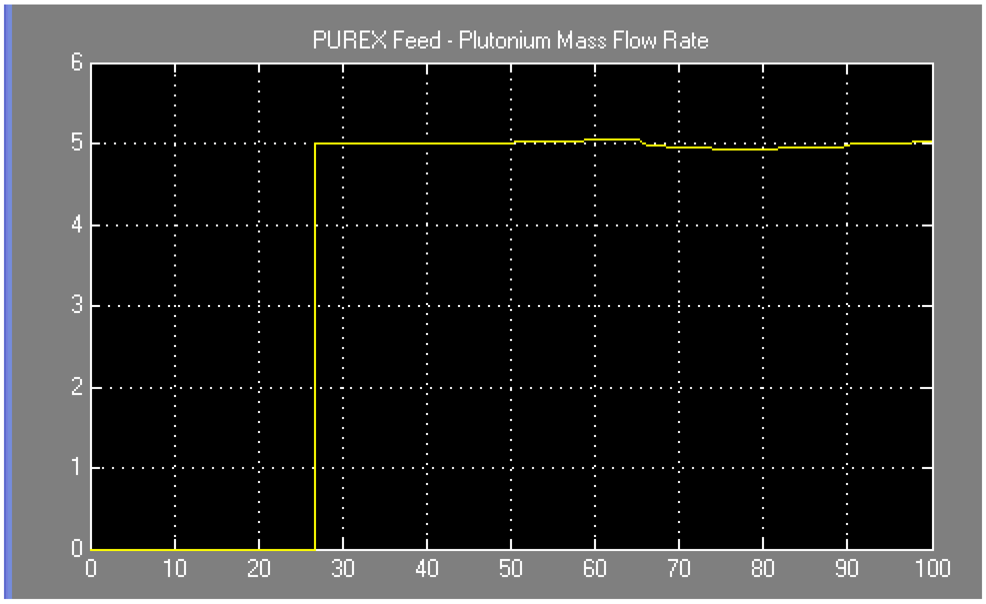


Figure 7: Transient test. Mass flow rate in kilogram per hour.

A third test was run to simulate a time-dependent plutonium diversion downstream of the solvent extraction unit starting at hour 40 and ending hour 60. In this case, the solvent extraction unit was invoked at intervals of 15 minutes. Figure 8 shows the results showing the diversion as a drop in the feed solution.



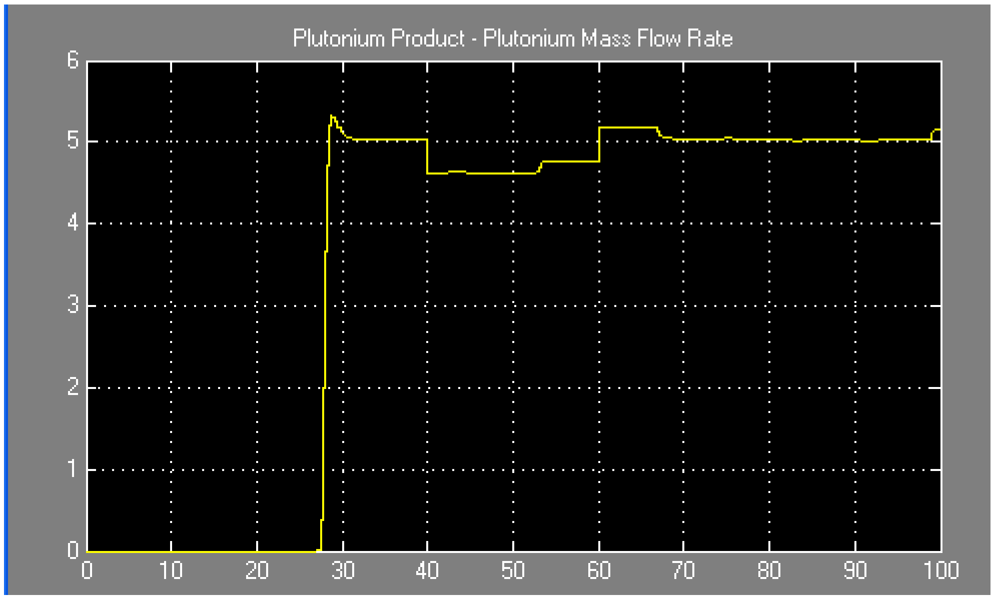


Figure 8: Plutonium diversion test; time-dependent diversion started at 40 h and ended at 60 h. Mass flow rate in kilogram per hour.

5.0 Discussion

This effort provided valuable experience toward the development of robust separations and safeguards models by the integration of transient process models.

The integration of transient solvent extraction simulation into the SSPM was successful, but the approach could be streamlined in the future. It should be possible to simplify the embedded files to make the coupling easier to follow and to prevent the depth of layers in the model. Running an underlying time-dependent solvent extraction simulation once per hour had only marginal impact on the resulting running time of the application. On a standard PC, the model runs at a rate of 1 simulated hour per second of CPU time.

A future goal of this work is to provide underlying models for the various unit operation processes in the plant. Alternatively, a plant-level framework used for plant design could incorporate the elements of safeguards design. The integration provides insights for future activities in the NEAMS program.

For the initial integration, the number of variables integrated between the two codes was kept limited to the feed rates, uranium concentration, and plutonium concentration. All of the other inputs necessary to define a solvent extraction process were assumed. For future work, additional variables like acid concentration and other operational variables of the solvent extraction process could also be integrated. This will require a more significant modification of the SSPM since acid concentration is currently not tracked in each stream. However, a structure is already in place for the user-defined variables to be pulled from the SSPM. In addition, development of the desired full functionality in the solvent extraction module for exploration of a wide range of transient process scenarios will require significant additional modifications to the code.

Beyond the upgrading of the current solvent extraction module to provide a full, modernized SEPHIS, there is significant opportunity for continued improvement of solvent extraction modeling for greater realism and functionality. Targets for future upgrading of solvent extraction modules include improved models of contactor types, improved thermodynamic models, and incorporation of mass transfer and reaction processes.

The transient solvent extraction simulation provides useful information about the amount of nuclear material held up in the PUREX contactors during normal operation. This amount is important to quantify for materials accountancy reasons. Now that the SSPM has taken a first important step toward having this more accurately modeled, new materials accountancy or process monitoring strategies can be developed that take the process holdup into account.

More extensive verification and validation will be required in future work. Validation has been performed in only a very limited case to roughly determine if the results match up to independent SEPHIS runs. Future efforts will need to outline a plan for comparing the results to actual data.

6.0 Conclusion

A transient solvent extraction simulation code has been successfully integrated into the SSPM safeguards model. This integration is a first step and can be expanded to integrate more variables and functionalities in future work. Preliminary results show that the new module driven by the SSPM is separating the uranium and plutonium streams as expected. It also calculates the correct time lag after transient conditions.

A much broader result is that this work provides a template for future code coupling. The SSPM model could be sent out to other countries, labs, universities, or industry where they could follow the template to couple their own proprietary separations codes. This work may also provide inputs for the design of a much larger plant simulation model that combines the capabilities of a number of different codes and allows for the Safeguards by Design approach.

7.0 References

1. Rainey, R. H., and S. B. Watson, 1975. *Modification of the SEPHIS Computer Code for Calculating the PUREX Solvent Extraction System*, Oak Ridge National Laboratory Report ORNL-TM-5123. Available at http://www.osti.gov/energycitations/servlets/purl/4167160-Ke5fTt/4167160.PDF

2. Groenier, W. S., R. H. Rainey, and S. B. Watson, 1979. “An analysis of the transient and steady-state operation of a countercurrent liquid-liquid solvent extractions process,” *Ind. Eng. Chem. Process Des. Dev.* **18** (3), 385-39.

3. B.B. Cipiti & F.A. Duran, “A Systems Engineering Process for Safeguards Design,” not yet published, Sandia National Laboratories (2009).

4. B.B. Cipiti, “Separations and Safeguards Performance Model (SSPM),” SAND2009-4896 (August 2009).

5. Groenier, W. S. *Calculation of the Transient Behavior of a Dilute Purex Solvent Extraction Process having Application to the Reprocessing of LMFBR Fuels*, ORNL-4746, April 1972

6. Marsh, D. L. Design of a Solvent Extraction Process to Separate Plutonium and Neptunium. University of Tennessee, 1998. M.S. Thesis.

7. Mitchell, A. D. *SEPHIS-MOD4: A User's Manual to a Revised Model of the Purex Solvent Extraction System*. Oak Ridge National Laboratory, 1979. ORNL-5471

8. Groenier, W. S. *Technical Manual for SEPHIS MOD4, Version 2.11*, Oak Ridge National Laboratory ORNL-TM-11589, February 1991.

9. Groenier, W. S. *User’s Manual for SEPHIS MOD4 Version 2.11*, Oak Ridge National Laboratory ORNL-TM-11588, February 1991.

10. Jubin, R. T. *A Modified Mathematical Model for Calculating Distribution Coefficients for U(VI), PU(IV), and Nitric Acid in the Uranyl Nitrate-Plutonium(IV) Nitrate-Nitric Acid-Water/Tributyl Phosphate System* Oak Ridge National Laboratory ORNL-TM-7217, April 1980

11. Jubin, R. T. *Revisions to the Mathematical Model Used in the SEPHS and MATEX Computer Codes for Calculating Distribution Coefficients of U(VI), Pu(IV) and Nitric Acid in the PUREX Chemical System Oak Ridge National Laboratory* ORNL-TM-9960, October 1986

Appendix

Embedded Function Code:

```
function [UProd,Solvent,PuProd,Raff] =
fcn(Feed,Flowrate,Uconc,PuConc,SolventFlow,ScrubFlow,StripFlow,NumStagesUREX,
VolFracTBP,Tempinit,Tempfeed,RatePUIV,TimeIncr,Stoptime,Tol,VolumeMixer,Nitri
cConc,PuRedConc,InNitrateConc,FeedStage,ScrubStage,StripStage)
% This block supports the Embedded MATLAB subset.
% The purpose of this embedded function is to call the Solvent Extraction
Unit interface (m-file).

eml.extrinsic('SolventExtractionUnit');

PUREXRaffGain=[1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;
1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1
;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;0;1;0;1;1;1;1;1;0.103;0.01];

persistent flagx;
uProdStreamUMassFlowRate  = 0;  % Set the mass flow rates to zero at startup
uProdStreamPuMassFlowRate = 0;  % Overwritten when Solvent Extraction Unit is
puProdStreamUMassFlowRate  = 0; % called, these fields are needed to produce
puProdStreamPuMassFlowRate = 0; % continuous outputs for Simulink

sIn.control.coldStart = false;  % The default is that the model is not at
                                  coldstart

sIn.control.runtimeLength_min = Stoptime;

if Flowrate>0
   sIn.uranium.concentration   = Uconc/Flowrate*1000;
   sIn.plutonium.concentration = PuConc/Flowrate*1000;
   % These fields pull the U and Pu concentration from the Simulink model
   sIn.uranium.concentrationUnit   = 'grams/liter';
   sIn.plutonium.concentrationUnit = 'grams/liter';
   if isempty(flagx)  % This argument sets coldstart to true only at startup
         sIn.control.coldStart = true;
         flagx=1;
   end
   % This field runs the Sovent Extraction Unit m-file
   sOut = SolventExtractionUnit(sIn);
   uProdStreamFlowRate  = sOut.uraniumProductStream.flowRate;
   uProdStreamUCc       = sOut.uraniumProductStream.uConcentration;
   uProdStreamPuCc      = sOut.uraniumProductStream.puConcentration;
   uProdStreamUMassFlowRate  = uProdStreamFlowRate * uProdStreamUCc;
   uProdStreamPuMassFlowRate = uProdStreamFlowRate * uProdStreamPuCc;

   puProdStreamFlowRate = sOut.plutoniumProductStream.flowRate;
   puProdStreamUCc      = sOut.plutoniumProductStream.uConcentration;
   puProdStreamPuCc     = sOut.plutoniumProductStream.puConcentration;
   puProdStreamUMassFlowRate  = puProdStreamFlowRate * puProdStreamUCc;
   puProdStreamPuMassFlowRate = puProdStreamFlowRate * puProdStreamPuCc;

end
```

```
Solvent = SolventFlow;
Raff = Feed.*PUREXRaffGain;
UProd = [zeros(1,91) uProdStreamUMassFlowRate*60/1000 0
uProdStreamPuMassFlowRate*60/1000 0 0 0 0 0 Flowrate*.3425 0];
PuProd = [zeros(1,91) puProdStreamUMassFlowRate*60/1000 0
puProdStreamPuMassFlowRate*60/1000 0 0 0 0 0 Flowrate*.3425 0];
% The previous fields use the results of the Solvent Extration Unit to
determine the outputs of the PUREX contactor bank
end
```

Solvent Extraction Unit Wrapper Octave/MATLAB code

```
%###########################################################################
%
% SOLVENT EXTRACTION UNIT module for Octave/MATLAB.
%
% Collaboration Sandia/ORNL for NA-242.
%
% This is a place holder for a full solvent extraction module written in Octave/MATLAB
% language. At the moment it performs some necessary preparation steps for a system
% call to run an external executable, reads its output, and return data to the calling
% program.
%
% The first call to this SolventExtractionUnit() function starts the unit.
% "Cold start" means the unit starts from zero concentration
% profile across the unit. The runtimeLength input variable sets the time length
% the unit will run for. The state of the unit is saved on a persisten data file
% and the next time the function is called the unit may start from this saved
% state if the user chooses to do so. The new starting conditions at the moment can
% only be given in terms of concentrations of uranium and plutonium in grams/L.
% A second call to the function with coldStart set to false will run the unit
% for an additional runtimeLength specified in input structure.
%
% This function expects a struct input as follows (the list will increase as
% the developement of the module progresses):
%
% sIn.control.coldStart = x;  (x is either true or false)
% sIn.control.runtimeLength_min = y; ( y is time in minutes; e.g. y = 1000.0)
%
% A companion driver code shows examples of how to define the input.
%
% This function returns a data structure. Inspecting the contents of the
% stucture will return self-explanatory information on the data. This choice
% of return value allows for flexibility in changing the internals of this
% function without affecting the calling program. However the calling program
% does have to operate on the returned data structure to obtain the real data.
% This can be done with support of native querying funtions. The companion
% driver code has examples.
%
% vfda::Valmor de Almeida; dealmeidav@ornl.gov
%
% Thu Sep  3 12:02:49 EDT 2009 scale up factor temporarily added; vfda
% Thu Sep  3 10:00:42 EDT 2009 output structrure data in terms of streams; vfda
% Wed Jun 17 15:01:15 EDT 2009 tested on linux and windows XP; vfda
% Tue Jun  9 21:43:11 EDT 2009 created; vfda
%#########################################################################
```

```
function sOut = SolventExtractionUnit( sIn )

 assert( isstruct(sIn) );

 coldStart     = getfield(sIn,'control','coldStart');
 runtimeLength = getfield(sIn,'control','runtimeLength_min');

 feedFlowRate     = getfield(sIn,'feedStream','flowRate');
 feedFlowRateUnit = getfield(sIn,'feedStream','flowRateUnit');
 assert(strcmp(feedFlowRateUnit,'liter/min'));

 uraniumCc     = getfield(sIn,'uranium','concentration');
 uraniumCcUnit = getfield(sIn,'uranium','concentrationUnit');
 assert(strcmp(uraniumCcUnit,'grams/liter'));

 plutoniumCc     = getfield(sIn,'plutonium','concentration');
 plutoniumCcUnit = getfield(sIn,'plutonium','concentrationUnit');
 assert(strcmp(plutoniumCcUnit,'grams/liter'));

%--------------------------------------------------------------------------------
% Scale up

 scaleUpFactor = 1.0;

 if ( feedFlowRate > 0.35 )
   scaleUpFactor = feedFlowRate / 0.35;
 end

%--------------------------------------------------------------------------------
% Write input file for external solvent extraction module

 inputFileName = 'input.dat';

 WriteInputFile( inputFileName, coldStart, runtimeLength, uraniumCc, plutoniumCc );

%--------------------------------------------------------------------------------
% Run solvent extraction module externally

 executableModuleName = 'purex.x';

 if ispc()
   command = executableModuleName;
   dos(command);
 elseif isunix()
   command = ['./',executableModuleName];
   unix(command);
 else
   error('Do not know what OS I am running on!\n Bailing out.\n');
 end

%--------------------------------------------------------------------------------
% Read Purex output

 fin = fopen('ornl.purex','r');

 s = fscanf(fin,' %s',7);

 for i = 1:4

  flowRate = str2num(fscanf(fin,' %s',1));
  uCc      = str2num(fscanf(fin,' %s',1));
  puCc1    = str2num(fscanf(fin,' %s',1));
  puCc2    = str2num(fscanf(fin,' %s',1));
  if (i==1)
    uProdStreamFlowRate = flowRate;
    uProdStreamUCc      = uCc;
    uProdStreamPuCc     = puCc1 + puCc2;
  end

  if (i==2)
    puProdStreamFlowRate  = flowRate;
    puProdStreamUCc       = uCc;
    puProdStreamPuCc      = puCc1 + puCc2;
  end

  if (i==3)
    raffinateStreamFlowRate = flowRate;
    raffinateStreamUCc      = uCc;
    raffinateStreamPuCc     = puCc1 + puCc2;
  end

  if (i==4)
    organicWasteStreamFlowRate = flowRate;
    organicWasteStreamUCc      = uCc;
    organicWasteStreamPuCc     = puCc1 + puCc2;
  end

 end

 fclose(fin);

%--------------------------------------------------------------------------------
% Fill in output structure data

% sOut.timing.cumulativeRuntime_min = cumulativeRuntime_min;

 sOut.uraniumProductStream.flowRateUnit       = 'liter/min';
 sOut.uraniumProductStream.flowRate           = uProdStreamFlowRate*scaleUpFactor;
 sOut.uraniumProductStream.uConcentrationUnit = 'grams/liter';
 sOut.uraniumProductStream.uConcentration     = uProdStreamUCc;
 sOut.uraniumProductStream.puConcentrationUnit = 'grams/liter';
 sOut.uraniumProductStream.puConcentration     = uProdStreamPuCc;

 sOut.plutoniumProductStream.flowRateUnit        = 'liter/min';
 sOut.plutoniumProductStream.flowRate            = puProdStreamFlowRate*scaleUpFactor;
 sOut.plutoniumProductStream.uConcentrationUnit  = 'grams/liter';
 sOut.plutoniumProductStream.uConcentration      = puProdStreamUCc;
 sOut.plutoniumProductStream.puConcentrationUnit = 'grams/liter';
 sOut.plutoniumProductStream.puConcentration     = puProdStreamPuCc;

 sOut.raffinateStream.flowRateUnit        = 'liter/min';
 sOut.raffinateStream.flowRate            = raffinateStreamFlowRate*scaleUpFactor;
 sOut.raffinateStream.uConcentrationUnit  = 'grams/liter';
 sOut.raffinateStream.uConcentration      = raffinateStreamUCc;
 sOut.raffinateStream.puConcentrationUnit = 'grams/liter';
 sOut.raffinateStream.puConcentration     = raffinateStreamPuCc;

 sOut.organicWasteStream.flowRateUnit        = 'liter/min';
 sOut.organicWasteStream.flowRate            =
organicWasteStreamFlowRate*scaleUpFactor;
 sOut.organicWasteStream.uConcentrationUnit  = 'grams/liter';
 sOut.organicWasteStream.uConcentration      = organicWasteStreamUCc;
 sOut.organicWasteStream.puConcentrationUnit = 'grams/liter';
 sOut.organicWasteStream.puConcentration     = organicWasteStreamPuCc;
```

```
%-------------------------------------------------------------------------------
% Clean up

% unlink('ornl.purex');
% unlink('input.dat');

 return;

end

%###############################################################################
% Helper functions
%###############################################################################

function WriteInputFile( fileName, coldStart, runtimeLength, uCc, puCc )

 fout = fopen(fileName, 'w');

 s = 'ORNL PuREx Solvent Extraction Processing Unit \n'; fprintf(fout, s);
 s = 'ornl\n'; fprintf(fout, s);
 s = strcat('\".\"','\n'); fprintf(fout, s);
 s = '48\n'; fprintf(fout, s);
 s = 'T\n'; fprintf(fout, s);
 s = '33\n'; fprintf(fout, s);
 s = '27\n'; fprintf(fout, s);
 s = ' 2\n'; fprintf(fout, s);
 s = ' 2\n'; fprintf(fout, s);
 s = '3\n'; fprintf(fout, s);
 s = '00000.30\n'; fprintf(fout, s);
 s = '000040.0\n'; fprintf(fout, s);
 s = ' \n'; fprintf(fout, s);
 s = '000000.1\n'; fprintf(fout, s);
 s = '000200.0\n'; fprintf(fout, s);
 s = sprintf('%8.1f\n',runtimeLength); fprintf(fout, s);
 s = '000.0001\n'; fprintf(fout, s);
 s = ' 1\n'; fprintf(fout, s);
 s = ' \n'; fprintf(fout, s);
 if (coldStart)
    s = 'F\n'; fprintf(fout, s);
 else
    s = 'T\n'; fprintf(fout, s);
 end
 s = 'F\n'; fprintf(fout, s);
 s = 'F\n'; fprintf(fout, s);
 s = ' \n'; fprintf(fout, s);
 s = ' \n'; fprintf(fout, s);
 s = ' \n'; fprintf(fout, s);
 s = ' 1       1      1.3      0.05                                  50.0
1   \n'; fprintf(fout, s);
 s = '17       1      0.6      0.05                      0.5     0.5     20.0
1   \n'; fprintf(fout, s);
 s = '32       0      0.5                                            20.0
1   \n'; fprintf(fout, s);
 s = '33       1      0.15     0.5                                   40.0
1\n'; fprintf(fout, s);
 s = '38       1      0.05     5.0                                   40.0
1\n'; fprintf(fout, s);
 s1 = '42       1      0.35     3.5    %8.1f%8.1f%25c 40.0     1\n';
 s = sprintf(s1,uCc,puCc,' ');
 fprintf(fout,s);

 s = '48       0      1.0                                            40.0
0\n'; fprintf(fout, s);
```

```
 s = ' \n'; fprintf(fout, s);
 s = ' \n'; fprintf(fout, s);
 s = ' \n'; fprintf(fout, s);
 s = '16       1      1.3         1       \n'; fprintf(fout, s);
 s = '32       1      0.6         0       \n'; fprintf(fout, s);
 s = ' \n'; fprintf(fout, s);
 s = ' \n' ; fprintf(fout, s);
 s = ' \n'; fprintf(fout, s);
 s = '    1        1    1.8                1        \n'; fprintf(fout, s);
 s = '    2        1    3.0                0        \n'; fprintf(fout, s);

 fclose(fout);

end
```

Distribution

1	John McClelland-Kerr (Department of Energy)
1	Rebecca Stevens (Department of Energy)
1	Terry Todd (Idaho National Laboratory)
1	James Bresee (Department of Energy)
1	Mike Miller (Los Alamos National Laboratory)
1	Brad Williams (Department of Energy)
1	Valmor de Almeida (Oak Ridge National Laboratory)
1	Ian Gauld (Oak Ridge National Laboratory)
1	Joseph Birdwell (Oak Ridge National Laboratory)
1	David DePaoli (Oak Ridge National Laboratory)
1	Mike Ehinger (Oak Ridge National Laboratory)
1	Robert Jubin (Oak Ridge National Laboratory)
1	Robert Wham (Oak Ridge National Laboratory)

1	Ben Cipiti, 6774
1	Ken Sorenson, 6774
1	John Kelly, 6770
1	Central Technical Files, 8944
1	Technical Library, 9536

Sandia National Laboratories